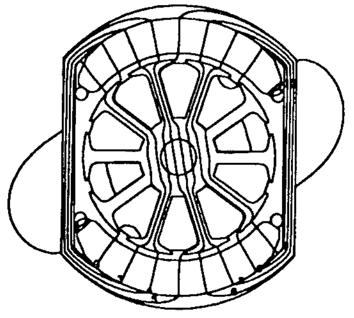
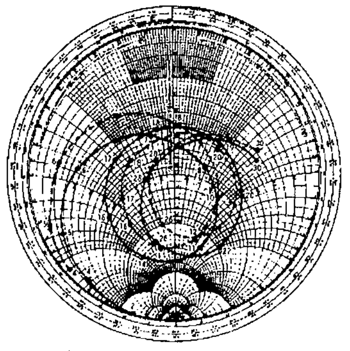
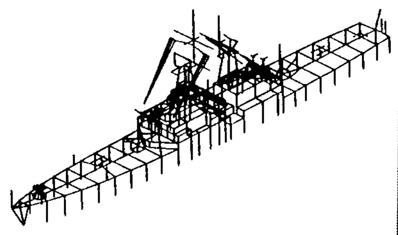
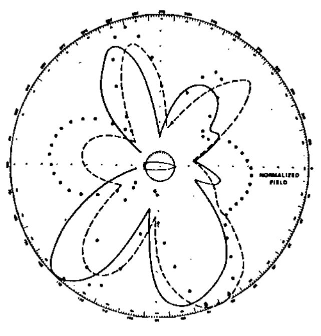
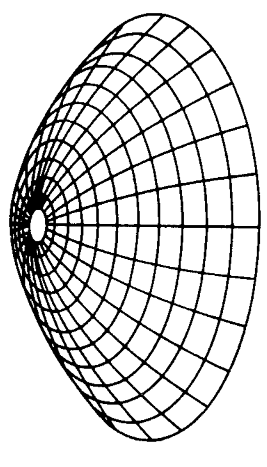







Applied Computational Electromagnetics Society Journal

Special Issue on Computational Electromagnetics and High-performance Computing

Guest Editors
David B. Davidson,
and
Tom Cwik

1998
Vol. 13 No. 2

ISSN 1054-4887

GENERAL PURPOSE AND SCOPE. The Applied Computational Electromagnetics Society Journal hereinafter known as the ***ACES Journal*** is devoted to the exchange of information in computational electromagnetics, to the advancement of the state-of-the-art, and to the promotion of related technical activities. A primary objective of the information exchange is the elimination of the need to "re-invent the wheel" to solve a previously-solved computational problem in electrical engineering, physics, or related fields of study. The technical activities promoted by this publication include code validation, performance analysis, and input/output standardization; code or technique optimization and error minimization; innovations in solution technique or in data input/output; identification of new applications for electromagnetics modeling codes and techniques; integration of computational electromagnetics techniques with new computer architectures; and correlation of computational parameters with physical mechanisms.

SUBMISSIONS. The ***ACES Journal*** welcomes original, previously unpublished papers, relating to **applied computational electromagnetics**.

Typical papers will represent the computational electromagnetics aspects of research in electrical engineering, physics, or related disciplines. However, papers which represent research in **applied computational electromagnetics** itself are equally acceptable.

Contributions may be sent to the Editors-in-Chief, Dr. Ahmed Kishk or Dr. Allen Glisson
Department of EE
University of Mississippi
University, MS, 38677 USA
Phone: 601-232-5385 (Ahmed)
Phone: 601-232-5353 (Allen)
Fax: 601-232-7231
email:ahmed@olemiss.edu
email:aglisson@mail.olemiss.edu. See "Information for Authors" on inside of back cover.

SUBSCRIPTIONS. All members of the Applied Computational Electromagnetics Society (**ACES**) who have paid their subscription fees are entitled to receive the***ACES Journal*** with a minimum of three issues per calendar year.

Visit us on line at: www.emclab.umr.edu/aces

Back issues, when available, are $15.00 each. Subscriptions to **ACES**, orders for back issues of the ***ACES Journal*** and changes of addresses should be sent to:

Dr. Richard W. Adler
ACES Executive Officer
ECE Department, Code ECAB
Naval Postgraduate School
833 Dyer Road, Room 437
Monterey, CA 93943-5121 USA

Allow four week's advance notice for change of address. Claims for missing issues will not be honored because of insufficient notice or address change or loss in mail unless the secretary is notified within 60 days for USA and Canadian subscribers or 90 days for subscribers in other countries, from the last day of the month of publication. For information regarding reprints of individual papers or other materials, see "Information for Authors".

LIABILITY. Neither ACES or the ***ACES Journal*** editors are responsible for any consequence of misinformation or claims, express or implied, in any published material in an***ACES Journal*** issue. This also applies to advertising, for which only camera-ready copies are accepted. Authors are responsible for information contained in their papers. If any material submitted for publication includes material which has already been published elsewhere, it is the author's responsibility to obtain written permission to reproduce such material.

APPLIED COMPUTATIONAL ELECTROMAGNETICS SOCIETY Journal

July 1998
Vol. 13 No. 2

ISSN 1054-4887

The ACES Journal is abstracted in INSPEC, in Engineering Index, and in DTIC.

The second, third, fourth, and fifth illustrations on the front cover have been obtained from Lawrence Livermore National laboratory.

The first illustration on the front cover has been obtained from FLUX2D software, CEDRAT S.S. France, MAGSOFT Corporation, New York.

THE APPLIED COMPUTATIONAL ELECTROMAGNETICS SOCIETY

JOURNAL EDITORS

THE APPLIED COMPUTATIONAL ELECTROMAGNETICS SOCIETY

SPECIAL ISSUE OF THE ACES JOURNAL ON COMPUTATIONAL ELECTROMAGNETICS AND HIGH-PERFORMANCE COMPUTING

Vol. 13 No. 2 July 1998

TABLE OF CONTENTS

TABLE OF CONTENTS (cont.)

Editorial — Special Issue of the ACES Journal on Computational Electromagnetics and High-Performance Computing

Guest editors:
David B Davidson (Univ. Stellenbosch, South Africa) and Tom Cwik (JPL, USA)

Introduction

This special issue contains thirteen papers, all dealing with aspects of high-performance computing (HPC) applied to computational electromagnetics. The majority of the papers address either integral equation or FD-TD formulations, with a few papers focusing specifically on fast algorithms in an HPC context. Readers will find it interesting to compare papers in the issue with previous special issues of other journals and monographs, for example [1, 2]. HPC issues first attracted significant attention in electromagnetics around a decade ago, with the present guest editors amongst the earlier workers in the field [3, 4].

The papers

In keeping with ACES traditions, the papers have a significant international flavour, with contributions from England, Hong Kong, Germany, Italy, Northern Ireland, South Africa and the USA.

Integral equation methods, especially the frequency domain formulation widely known in CEM as the Method of Moments (MoM), continue to play a very important role in practical CEM, not least due to the availability of very powerful and user-oriented codes such as NEC. Five papers in this special issue consider HPC implementations of integral equation formulations — three consider various hybridisations of the (frequency domain) MoM with other techniques, one discusses a parallel implementation of NEC2 and the fifth consider a time domain integral equation formulation. The paper by Jakobus describes a powerful MoM code (FEKO) which includes hybrid treatments using physical optics and/or the uniform theory of diffraction. He discusses the performance of FEKO in both distributed workstation (IBM RS6000) and (potentially massively) parallel processor (Intel Paragon and CRAY-T3E) environments. With hybrid codes, the matrix solve does not necessarily dominate the MoM solution time; furthermore, the use of special Green's functions may result in either matrix fill or field computation dominating the run-time and Jakobus' paper addresses these issues in the context of a number of important practical problems. Similar experiences with another hybrid MoM code are reported by Shen et al. using an IBM SP-2 platform for an asymptotic phase method code; this formulation is related to the physical optics approach reported by Jakobus. The same SP-2 platform is used by Nitch et al. in their paper, which discusses their parallel implementation of the widely-used public domain MoM code NEC2. Smelyanskiy et al. discuss another hybrid MoM code (in this case combining the MoM with a modal expansion to handle propagation through the inlet duct of a jet engine) on a CRAY-C90 machine. The paper by Dodson et al. discusses optimisation issues for *time* domain integral equation formulations; the hardware in this case was a DEC Alpha workstation.

The popularity of the Finite-Difference Time-Domain (FD-TD) method (at least in research environments) is clearly shown in this Special Issue. The standard (non-curvilinear) FD-TD method is inherently highly parallel and this is reflected by no less that four papers on the topic (and another uses the closely related Transmission Line Method). The paper by Mazumdar et al. discusses performance modelling in a variety of HPC environments, including the Silicon Graphics Power Challenge and Sun workstations networks. The paper by Thomas et al. discusses a powerful user-oriented FD-TD package for especially high-speed circuit simulation; results on a CRAY Origin 2000 system are reported. The paper by Mullen et al. addresses the FD-TD method in a distributed processing environment using standard Unix workstations. It presents a very readable description of the PVM harness, and considers the problems resulting from load imbalance in this type of environment. Stothard and Pomeroy discuss special purpose hardware for a Transmission Line Method (TLM) implementation. Their paper addresses Application Specific Integrated Circuits, specifically Field

Programmable Gate Arrays — FPGA. Excell's paper considers a number of issues of porting a FD-TD code into a parallel environment, in particular the Kendall Square Research KSR-1.

Fast algorithms have attracted considerable attention in the CEM community, since they hold out the tantalising promise of greatly reducing the operation count. Three papers in the Special Issue address this topic, concentrating on algorithmic issues. Caproni et al. discuss the use of closed-form Green's functions for stratified media problems, reporting order-of-magnitude improvements in a traditionally computationally expensive method. Brauer describes a finite element method which reliably extracts the eigenmodes of low-loss devices, and then uses these as basis functions for further multi-frequency analysis — in particular with filters, large numbers of frequency points are often required for adequate resolution. Again, order-of-magnitude speed-ups are reported. Finally, Liu et al. describe work using the measured equation of invariance for scattering by very large cylinders — the authors anticipate speed-ups by two to three orders-of-magnitude for specific problems.

Algorithms

As mentioned above, it is notable that the majority of the papers in this issue use either the FD-TD or the MoM. Conspicuously absent are papers addressing specifically finite element formulations (FEM) and HPC — the emphasis of Brauer's paper is on algorithmic improvements rather than parallelisation aspects of the FEM. This may present a slightly skewed picture of the current situation, but it does reflect the difficulty of parallelising calculations arising from the irregular mesh associated with the FEM. Also absent are papers on asymptotic methods such as the UTD (apart from Jakobus's discussion in the context of hybrid methods). The ray-tracing required by this type of formulation is inherently highly parallel; we speculate that asymptotic codes make less formidable demands on computational resources, and that this is the reason for the dearth of such papers in this issue. (Another factor may be that all the discrete, full-wave methods — FD-TD, MoM, FEM, TLM — all require prodigious amounts of memory, and this fact, as much as run-time, is what often drives developers to HPC implementations. Asymptotic methods are generally far less memory intensive).

A noteworthy aspect of the work reported in this issue is that no new specifically parallel algorithms have arisen in computational electromagnetics. Around a decade back, when parallel computing first attracted serious interest, there was speculation in some quarters that the rise of massively parallel computers would trigger entirely new algorithms that were only feasible in massively parallel computing environments. With hindsight, such claims appear as primarily marketing "hype". All the formulations used in this special issue work equally well (albeit more slowly) on "sequential" systems, and were originally derived for such systems.

Hardware

When HPC first came to the attention of the CEM community, it was often accompanied by highly specialised hardware, frequently purpose-built (as an example, see [4], where a custom-built transputer array was used). Apart from one paper considering special purpose processors as putative processing elements (FPGA's), all the other papers use relatively mainstream environments, reflecting a degree of maturity in the field. It is also notable that the old SIMD-MIMD classification has fallen away completely — HPC environments now are generally classified either as SMP (symmetric multi-processor), MPP (massively parallel processor) or distributed processing environments. The first is currently epitomised by systems such as the (Silicon Graphics) Origin; the number of processors is typically fairly modest, but memory is essentially shared; the second is epitomised by the (IBM) SP-2 and (CRAY) T3-E, with a large number of processors accessing distributed memory; and the last by heterogeneous networks of standard workstations, again with distributed memory but much slower communication networks than the purpose-built ones incorporated into MPP's. (The T3-E actually combines elements of both SMP and MPP paradigms, since is also contains a globally addressable memory subsystem).

Also noticeable has been the shake-out in manufacturers in the HPC sector. Thinking Machines Corp. and their Connection Machines (CM-2 and CM-5) are gone — indeed, no papers in this special issue report results on Connection Machine systems. Kendall Square Research is also no longer a commercial vendor (although one paper uses the KSR-1, which had some innovative features, not least a physically distributed memory which was accessed as shared memory by application programs, using a system of multi-level caches). CRAY Research/Silicon Graphics Inc. remain arguably the most influential commercial vendor in this field at the time of writing, although companies such as IBM and Hitachi continue to be involved. (Several papers report work on the IBM SP-2). Transputers have disappeared entirely from the HPC horizon, due partly to massive speed improvements in other RISC and CISC processors (DEC Alpha's, MIPS R10 000 and the latest

Pentiums) and partly due to a failure by the vendors to adequately support FORTRAN and C/C++.

Languages, compilers, system software and libraries

Although not specifically addressed by most authors, only two languages are serious contenders in the HPC environment: FORTRAN, in particular the latest variants such as FORTRAN 90 and 95; and C in its various forms. Development environments such as MATLAB are widely used in CEM, in particular for code prototyping, but are not currently routinely used for HPC applications.

Few authors explicitly address the issue of operating systems, but almost all the HPC environments in this special issue use Unix (in its many commercial variants) as the O/S. It seems likely that only Unix and future Windows variants are likely to survive significantly into the new millennium — although predictions involving the future are always risky, even more so with computing!

The rise of Linux, the public-domain Unix implementation, is also not addressed explicitly, but at the time of writing this editorial interesting work was in progress using cheap Linux-based systems to implement fairly tightly coupled distributed processing environments (the Beowolf project).

On reflection, the usability of current high-performance computing platforms remains disappointing: a major bottleneck has been the inadequacy of parallel compilers and system software. Although considerably improved over the systems of a decade ago (where such system software was sometimes entirely absent), fundamental items such as parallel I/O and easy-to-use parallel debuggers have not appeared. Better systems could greatly assist the user in porting code to the new architectures; the learning curve for current systems remains intimidatingly high for many potential users. Therefore, much of the high-performance activity is left to the few who want to do it and are paid to do it!

What is encouraging has been the emergence of two standardised "harnesses" — Parallel Virtual Machine (PVM) and Message Passing Interface (MPI). These provide standardised high-level communication routines (via libraries) to route data between processes, removing — or at least greatly reducing — the hardware dependent implementations which characterised earlier work [4].

The future of CEM and HPC

Our community has grown accustomed to what are actually astonishing improvements in processing speed throughout the last decade — in 1990, 100 MHz was a very fast clock speed indeed for desktop workstations and PC's, whereas there is reason to believe that the 1 GHz barrier will be reached by affordable RISC and/or CISC CPU's by the year 2000. There is currently speculation that pushing clock speeds significantly beyond 1GHz may prove a major technological challenge, but given the micro-electronic industry's success in overcoming previous "barriers", such speculation needs to be treated with caution. CPU's have also become very complex, with elaborate pre-emptive multi-tasking strategies and deep pipelines becoming standard on both RISC and CISC chips — indeed, it is questionable whether the RISC/CISC dichotomy still exists. It is also becoming increasingly clear that for many applications, raw processing speed is less significant than the ability to move data rapidly to and from memory — indeed, systems such as the SGi Origin range use R10 000 processors identical to those of their lower-end workstations, but obtain greater computational throughput by providing considerably higher processor/memory bandwidth.

Should the trend to faster CPU's finally slow down, CEM is now in a much stronger position to exploit parallel and/or distributed processing than was the case a decade ago. As reported in this Special Issue, there is now considerable expertise and experience in this field, as well as maturing software technologies to make implementation far less painful than was the case for early researchers in this field, although as commented previously, there is still scope for major improvements in this regard. "Sequential" computer technology has almost fifty years of development supporting it, whereas high-performance computing has a much shorter (ten to fifteen years) history.

The use of distributed processing, where the computational load is spread across a potentially heterogeneous network of workstations, using a standard network (eg. Ethernet) for communication, can be expected to increase — irrespective of improvements in processor speed — for "embarrassingly parallel" applications, such as a frequency loop in a frequency domain code. (Either PVM or MPI is generally used at present to implement the communication in such a distributed processing environment.)

For those who wish to venture into the field, the larger memories and faster total processing power supplied by modern high-performance computers allow problems to be solved with greater fidelity and with potentially much shorter turn-around times. If the problem is sufficiently large electrically, it enables the problem to be solved at all. In a competitive marketplace, it is these points that justify the extra effort necessary in using current architectures.

In closing, we would like to sincerely thank the

authors for their contributions. The deadlines were fairly tight and their cooperation in meeting them is highly appreciated. Furthermore, our heartfelt thanks to the reviewers for the Special Issue; all papers in this issue were reviewed by at least two reviewers, and the majority received three reviews. A list of reviewers concludes this editorial. Finally, we acknowledge our respective employers for support — in particular secretarial services — provided during the editorial process.

References

[1] Guest editor: V.F.Fusco, "Special issue on parallel and distributed processing techniques for electromagnetic field solution," *The International Journal of Numerical Modelling: Electronic Networks, Devices and Fields*, vol. 8, May–August 1995.

[2] Editors: T. Cwik, J.Patterson, *Computational Electromagnetics and Supercomputer Architecture.* PIER7 Progress in Electromagnetics Research, Cambridge, MA: EMW Publishing, 1993.

[3] R. H. Calalo, T. A. Cwik, W. A. Imbriale, N. Jacobi, P. C. Liewer, T. G. Lockhart, G. A. Lyzenga, and J. E. Patterson, "Hypercube parallel architecture applied to electromagnetic scattering analysis," *IEEE Trans. Magnetics*, vol. 25, pp. 2998–2990, July 1989.

[4] D. B. Davidson, "A parallel processing tutorial," *IEEE Antennas Propagat. Magazine*, vol. 32, pp. 6–19, April 1990.

Reviewers for the Special Issue

Gerald J Burke, Lawrence Livermore National Lab., USA
Robert J Burkholder, Ohio State Univ., USA.
Vaughn Cable, Lockheed-Martin, USA
Steve Castillo, New Mexico State Univ., USA.
Chris Christopolous, Univ. Nottingham, England.
Tom Cwik, JPL, USA.
David B. Davidson, Univ. Stellenbosch, South Africa.
Peter Excell, Bradford Univ., England.
Vincent F. Fusco, Queen's Univ., Northern Ireland.
Gary Haussman, Univ. Colorado, USA.
Richard C. Hall, Ansoft Corp., USA.
Bijan Houshmand, JPL, USA.
Steven Gedney, Univ. Kentucky, USA.
Kueichien Hill, Wright-Patterson AFB, USA
Nathan Ida, Univ. of Akron, USA.
Ulrich Jakobus, Univ. Stuttgart, Germany.
Vahraz Jamnejad, JPL, USA.
Dan S. Katz, JPL, USA.
A. Kishk, Univ. Mississippi, USA.
Joe D. Kotulski, Sandia National Lab., USA.
Frans J.C. Meyer, EMSS, South Africa
Derek A. Nitch, Univ. Witwatersrand, South Africa.
D. Hugo Malan, Cambridge Univ., England.
Chris Trueman, Concordia Univ., Canada.
Petrie Meyer, Univ. Stellenbosch, South Africa.
Andrew F. Peterson, Georgia Tech, USA.
Chris Repesh, Mission Research Corp., USA.
Neil Simons, Communications Research Centre, Canada.
C. David Turner, Sandia National Lab., USA.
Simon Walker, Imperial College, England.
David M. Weber, Univ. Stellenbosch, South Africa.
John W. Williams, SAIC, USA.
Ke-Li Wu, Com Dev Ltd, Canada.
Ming Yu, Com Dev Ltd, Canada.
Cinzia Zuffada, JPL, USA.

Application of Integral Equation and Hybrid Techniques to the Parallel Computation of Electromagnetic Fields in a Distributed Memory Environment

Ulrich Jakobus

Institut für Hochfrequenztechnik, University of Stuttgart,
Pfaffenwaldring 47, D-70550 Stuttgart, Germany
E-Mail jakobus@ihf.uni-stuttgart.de

ABSTRACT. **This paper describes the parallelization of the method of moments and hybrid code FEKO for execution on massively parallel supercomputers with a distributed memory as well as on clusters of connected workstations. The parallel implementation of the different phases of the solution process, e.g. matrix fill, solution of the system of linear equations, and near– and far–field computation is discussed in detail. Several results for different applications are given and the achieved performance is presented.**

1 Introduction

The method of moments (MoM) originally developed by Harrington in 1968 [1] remains a very popular technique for solving numerous electromagnetic radiation and scattering problems involving metallic or dielectric scattering bodies.

As compared with other numerical techniques such as the finite difference (FDTD) or finite element method (FEM), the main advantage lies with the fact that for most MoM formulations only boundaries must be discretized, i.e. instead of a three–dimensional discretization for FDTD or FEM and other techniques, only a two–dimensional discretization of boundaries is necessary for 3D problems. Thus the number of unknowns for the MoM is usually a magnitude smaller when compared to FDTD or FEM. However, the matrix is not sparse but dense so the memory requirement is rather critical, especially for higher frequencies.

In this paper, we consider the computer code FEKO, which has been developed at the University of Stuttgart and represents a rather comprehensive MoM and hybrid implementation. FEKO has been adapted such that it is capable of executing in parallel on massively parallel supercomputers as well as on clusters of connected workstations with a large distributed memory. In such a distributed memory environment, every node (e.g. workstation) has its own local memory which is not accessible by the other nodes as opposed to the shared memory concept where all processes have access to a global bank of memory. The terms node, processor, or processing element (PE) are used interchangeably in the following.

A general overview of parallel processing techniques in the area of computational electromagnetics (CEM) can be found in [2, 3]. The parallel FEM is described in [4, 5] and a parallel implementation of FDTD is presented in [6]. Asymptotic high frequency methods are considered in [7, 8]. Several authors have already described a parallel MoM formulation [9, 10, 11, 12, 13, 14, 15], but in this paper we put special emphasis also on achieving reasonable performance on heterogeneous clusters of workstations by implementing a dynamic load balancing scheme. FEKO also offers some hybrid extensions for higher frequencies and special geometries, and the parallelization of these parts is also considered. The hybrid extensions allow a reduction of the matrix size, so for the hybrid techniques high performance computing (HPC) is mainly used to reduce the computation time. Only when the conventional MoM is applied (e.g. for validation of the hybrid results), we quite often face problems with up to 40000 unknowns or even more, and then HPC is a must.

Our parallel implementation of FEKO is based on the *Message–Passing Interface* (MPI) standard [16], to which an excellent introduction can be found in [17]. Alternatively PVM *(Parallel Virtual Machine)* might be used as well. For massively parallel supercomputers (the results in this paper were obtained on an Intel Paragon and a CRAY T3E) the MPI functions and subroutines are usually provided in an optimized library by the manufacturer. But especially for clusters of connected workstations some public domain implementations are also available. We have compared MPICH [18] and LAM *(Local Area Multicomputer)*. MPICH was chosen for performance and compatibility reasons.

The general structure of FEKO including the hybrid extensions is presented in Section 2. Section 3 describes in detail the parallelization of the different phases of the solution process for the MoM part of FEKO, while Section 4 deals with the parallelization of the hybrid extensions. Finally, some results demonstrating the performance of the parallel implementation are presented in Section 5.

2 General structure of the sequential version of FEKO

2.1 Conventional MoM

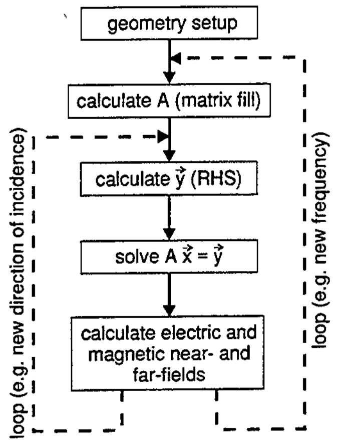

geometry setup
calculate A (matrix fill)
calculate y (RHS)
solve A x = y
calculate electric and magnetic near- and far-fields
loop (e.g. new direction of incidence)
loop (e.g. new frequency)

Fig. 1: Simplified structure of the MoM part of FEKO.

The simplified flow chart of the MoM part of FEKO is shown in Fig. 1. Some additional blocks must be included when considering hybrid extensions as described in Section 2.2.

The first block in Fig. 1 entitled *geometry setup* includes the allocation of memory, reading the geometrical data from an input file, the setup of the geometry and some optional error checks.

After this, the system matrix A is computed. Let N be the number of unknowns, then the CPU–time requirement for this phase is proportional to N^2. After the computation of the elements of the right–hand side (RHS) vector $\vec{y}$, the system of linear equations $A\,\vec{x} = \vec{y}$ can be solved resulting in the current vector $\vec{x}$. For very large matrices this step dominates the CPU time since there is an N^3 dependency. Once the currents are known, near- and far–fields can be computed. Several loops are possible as indicated in Fig. 1 by the dashed arrows.

The computer code FEKO supports metallic surfaces and wires as well as dielectric bodies. For the latter, two different formulations based on the surface and volume equivalence principle, respectively, have been implemented. Different basis functions $\vec{f}_n^i$ are involved:

- Linear roof–top basis functions $\vec{f}_n^1$ according to Rao, Wilton and Glisson [19] defined on flat triangular patches are used for metallic surfaces.
- Along metallic wires, overlapping triangular basis functions $\vec{f}_n^2$ are employed to represent the line current.
- Special basis functions $\vec{f}_n^3$ are required to model a current flow from metallic wires to surfaces at attachment points.
- An application of the surface equivalence principle to homogeneous dielectric scattering bodies leads to equivalent electric and magnetic surface current densities $\vec{J}$ and $\vec{M}$, respectively. Similar to metallic scattering problems, the surface of dielectric bodies are also subdivided into electrically small triangular patches. Basis functions $\vec{f}_n^4$ identical to $\vec{f}_n^1$ are used to represent $\vec{J}$, and for the magnetic surface current $\vec{M}$ a new set of vectorial basis functions $\vec{f}_n^5$ have been developed in Ref. [20].
- For inhomogeneous dielectric bodies the volume equivalence principle can be used in FEKO. The dielectric volume is subdivided into non–uniform cuboidal cells and within each cell three pulse basis functions $\vec{f}_n^6$ and three pulse basis functions $\vec{f}_n^7$ are employed in the three coordinate directions. $\vec{f}_n^6$ represents the equivalent electric volume current density and is required only for regions where the permittivity ε is not identical to the free space permittivity ε_0. The magnetic current is expanded into basis functions $\vec{f}_n^7$, and these basis functions are required only when $\mu \neq \mu_0$.

In addition to these 7 different basis functions $\vec{f}_n^i$, $i = 1 \ldots 7$, corresponding weighting functions $\vec{w}_m^i$ must be defined in order to convert the set of coupled integral equations into a system of linear equations. In general, we use the Galerkin procedure with $\vec{w}_m^i = \vec{f}_n^i$.

Fig. 2 shows the general structure of the MoM matrix A. The 7×7 sub-matrices A_{ij} associated with the combinations of basis functions $\vec{f}_n^j$ and weighting functions $\vec{w}_m^i$ can easily be identified. Note, however, that usually for an application to general radiation and scattering problems only some of these sub-matrices will be present. For most of the metallic problems, the submatrix A_{11} will be dominant in size, i.e. $N_1 \gg N_2, N_3$, and $N_4 = N_5 = N_6 = N_7 = 0$.

2.2 Hybrid extensions

FEKO is not only a pure MoM code, but includes some hybrid features allowing an efficient application also to high–frequency problems. The general idea is to reduce the number of unknowns N by applying a coupling of

$$
N \left\{ \begin{array}{ccccccc|c}
\vec{f}_n^1 & \vec{f}_n^2 & \vec{f}_n^3 & \vec{f}_n^4 & \vec{f}_n^5 & \vec{f}_n^6 & \vec{f}_n^7 & \\
\hline
A_{11} & A_{12} & A_{13} & A_{14} & A_{15} & A_{16} & A_{17} & \vec{w}_m^1 \\
A_{21} & A_{22} & A_{23} & A_{24} & A_{25} & A_{26} & A_{27} & \vec{w}_m^2 \\
A_{31} & A_{32} & A_{33} & A_{34} & A_{35} & A_{36} & A_{37} & \vec{w}_m^3 \\
A_{41} & A_{42} & A_{43} & A_{44} & A_{45} & A_{46} & A_{47} & \vec{w}_m^4 \\
A_{51} & A_{52} & A_{53} & A_{54} & A_{55} & A_{56} & A_{57} & \vec{w}_m^5 \\
A_{61} & A_{62} & A_{63} & A_{64} & A_{65} & A_{66} & A_{67} & \vec{w}_m^6 \\
A_{71} & A_{72} & A_{73} & A_{74} & A_{75} & A_{76} & A_{77} & \vec{w}_m^7 \\
\hline
N_1 & N_2 & N_3 & N_4 & N_5 & N_6 & N_7 &
\end{array} \right.
$$

Fig. 2: Matrix A consisting of 7×7 sub-matrices A_{ij} of different size.

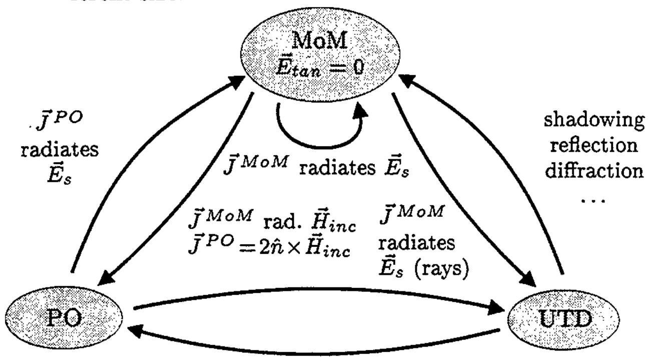

MoM
$\vec{E}_{tan} = 0$
$\vec{J}^{PO}$ radiates $\vec{E}_s$
shadowing
reflection
diffraction
...
$\vec{J}^{MoM}$ radiates $\vec{E}_s$
$\vec{J}^{MoM}$ rad. $\vec{H}_{inc}$
$\vec{J}^{PO} = 2\hat{n} \times \vec{H}_{inc}$
$\vec{J}^{MoM}$ radiates $\vec{E}_s$ (rays)
PO
UTD

Fig. 3: Hybrid method combining MoM, PO and UTD.

the MoM with high–frequency techniques such as Physical Optics (PO) with correction terms [21, 22, 23], Fock currents [24], or diffraction theory (UTD) [25]. The different coupling mechanisms are illustrated in Fig. 3.

Some examples considered in the references cited above demonstrate the drastic reduction of memory requirement of the hybrid technique as compared to an application of the MoM alone. There are, of course, some additional computations introduced by the coupling mechanisms depicted in Fig. 3, so that the flow chart of FEKO in Fig. 1 becomes more complicated, nevertheless there is also a considerable reduction in the CPU–time achievable.

Another hybrid extension of FEKO is the possible replacement of the free–space Green's function by a special Green's function for a homogeneous, layered dielectric sphere [26]. This technique can be used to optimize the performance of mobile telecommunications antennas radiating close to the operator's head. Only the antenna structure with the case must be discretized, hence the number of unknowns N can be kept relatively small.

3 Parallel implementation of the MoM in FEKO

3.1 Geometry setup

As briefly described in Section 2.1, the part of this phase of the solution process consuming the most time is the search for connected wires or edges between triangular patches or connections between wires and surface elements etc. Some acceleration techniques such as the spatial partitioning technique borrowed from computer graphics [27] together with boxing algorithms have been implemented in order to avoid, for example, the comparison of two triangular patches which are located far away. Hence, for large structures, the CPU time spent in this phase is rather small when compared to the matrix fill, matrix solve and field calculations. Therefore, after the allocation of memory (parallel by all processes), the input file is read only by one process. After setting up the geometry, the data are sent to all other processes using the `MPI_TYPE_STRUCT`, `MPI_TYPE_COMMIT`, `MPI_BCAST` and the `MPI_TYPE_FREE` subroutines. Using these subroutines enables us to send whole Fortran COMMON blocks which is much more efficient than sending single variables or arrays.

As an option it has also been implemented to preprocess the geometry on a PC or workstation. The connection information and other computed data are written to a file, which can be read later by the parallel job together with the original input file.

3.2 Matrix fill

When trying to parallelize the setup of the matrix, i.e. the computation of matrix elements a_{mn}, $m, n = 1 \ldots N$ with $N = \sum_{i=1}^{7} N_i$ (see Fig. 2), special care must be taken in order to preserve the advantages of a possible symmetry of the structure or of some efficient matrix fill techniques. Exploiting symmetry allows to reduce the number of unknowns from N to a smaller value $\tilde{N}$. If the available main memory permits, then in a first step an $\tilde{N} \times N$ matrix is constructed which is later compressed to a square $\tilde{N} \times \tilde{N}$ matrix. The advantage of this procedure is that we can use some symmetry relations between the matrix elements in a row to further reduce the CPU–time. However, if not enough main memory is available for the $\tilde{N} \times N$ matrix, it is also possible to directly use only an $\tilde{N} \times \tilde{N}$ array at the expense of a slightly increased CPU–time.

Different storage schemes for dense matrices, to the effect that the partitions can be assigned to different processors, are described e.g. in [28, Section 5]. When exploiting symmetry, when using efficient matrix fill techniques, or when using hybrid techniques where a modification of the matrix elements is necessary because of the coupling between MoM– and asymptotic region, it is is highly advantageous to keep a whole row of the matrix on one node.

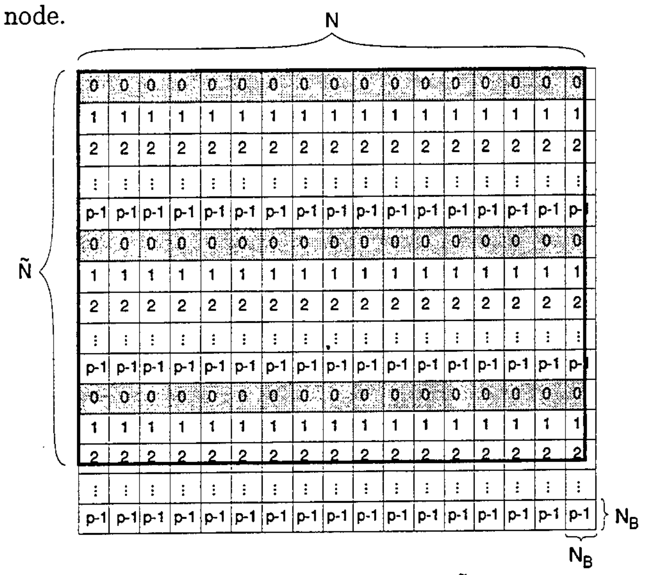


Fig. 4: Distributed storage scheme of the $\tilde{N} \times N$ matrix on p nodes (the numbers in the blocks indicate the nodes).

Hence, for a parallel computation and a distributed storage of the elements a_{mn}, the resulting rectangular matrix with $\tilde{N}$ rows and N columns is split among the p processors according to Fig. 4 in a one–dimensional block–cyclic row distribution. One square block consists of $N_B \times N_B$ elements with a block size in the range $1 \leq N_B \leq \lceil \frac{\tilde{N}}{p} \rceil$ (the notation $\lceil x \rceil$ represents the smallest integer value that is greater than or equal to x), see below (especially Fig. 10) for some considerations concerning an optimal choice of N_B.

The CPU time required for the computation of a single matrix element differs over a wide range by a factor of about 10 or even more, since various adaptive integration and testing schemes are applied depending on the distance $|\vec{r} - \vec{r}\,'|$ of source and observation point $\vec{r}\,'$ and $\vec{r}$, respectively. Also in some cases when $\vec{r}$ is located within the integration domain or very close to it, singular or quasi singular integrals are evaluated analytically. But on average, the time required to compute an element a_{mn} of the sub-matrix A_{11} (see Fig. 2) on an IBM RS 6000 workstation is in the range of $120 \ldots 250\,\mu$s. Experience has shown that the CPU time to compute the matrix elements in different blocks of N_B rows and N columns of the sub-matrix A_{11} is about the same (averaging effect).

However, we cannot expect this any more when different sub-matrices are involved. For a matrix row with weighting functions $\vec{w}_m^1$ (sub-matrices A_{1j}) the required CPU–time is most likely not equal to the CPU–time for a matrix row with different weighting functions $\vec{w}_m^j$, $j = 2 \ldots 7$. To overcome this problem, we have introduced a special mapping function allowing us to exchange matrix rows when the original matrix according to Fig. 2 is mapped to the matrix underlying the distributed storage scheme in Fig. 4. The mapping function is constructed so that each node contains about the same number of rows from all the sub-matrices, see Section 5 for an example (Figs. 12 and 13).

Another possible drawback of the described parallel matrix filling technique is that it is not very easy to implement a dynamic load balancing scheme. In the startup phase of the parallel version of FEKO, when the values of N, $\tilde{N}$ and the number of nodes p is known, a suitable block size N_B is selected and remains fixed. On a cluster of connected workstations, where the different processors are not exclusively assigned to a certain user, it might happen, that during the computation of the matrix elements another user starts some jobs on one of the workstations and, consequently, the parallel job on this node takes much longer than on the other workstations. There is currently no way to react to this situation, i.e. the other nodes have to wait (barrier) until the slowest machine has also finished its calculations. We have some ideas on how to overcome this problem, e.g. the faster nodes calculate some more matrix elements which were initially assigned to the slower machine and will then send them as a whole block to the slower machine. But this extension has not yet been implemented.

Note that this problem does not exist on massively parallel supercomputers where the different PEs are all of the same speed and the nodes are exclusively assigned to one single process. There are only some small differences in the required CPU–time on the different nodes due to the fact that we use adaptive integration techniques with a variable number of integration points or that the elements of the different sub-matrices A_{ij} are based on different equations, see Figs. 11, 12 and 13 below in Section 5 for some performance results.

3.3 Solution of the system of linear equations

A review of parallel matrix solvers for CEM applications is given in [29], the paper [30] concentrates on a parallel LU algorithm.

As already mentioned in the introduction, the whole parallelization of FEKO is based on the MPI standard. Several portable linear algebra packages based on MPI exist and their suitability has been compared.

PIM [31] provides iterative solvers such as CG, BiCG, BiCGStab, QMR, or GMRES. However, convergence studies with the sequential version of FEKO have shown that convergence is only satisfactory for some dielectric structures treated with the volume equivalence principle (sub-matrices A_{66}, A_{67}, A_{76}, A_{77} in Fig. 2). For metallic structures, which in contrast to the Fredholm integral equation of the second kind for these dielectric problems is based on a Fredholm integral equation of the first kind (EFIE), convergence is rather poor. However, some recent publications (e.g. [32, 33, 34]) show, that by using suitable preconditioners the convergence of the above mentioned iterative schemes can be improved significantly.

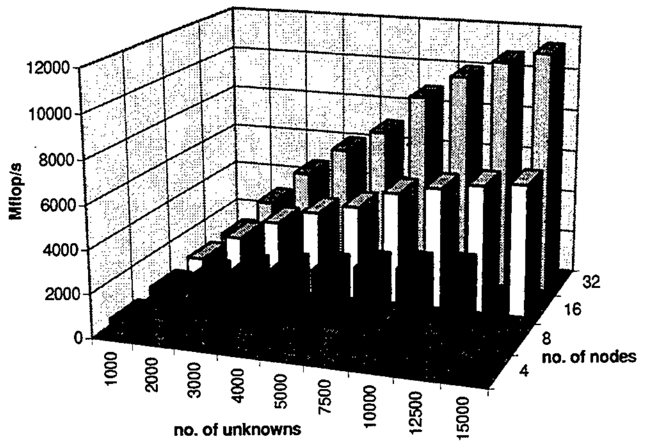


Fig. 5: Performance of the LU factorization on the CRAY T3E using ScaLAPACK (data from [35]).

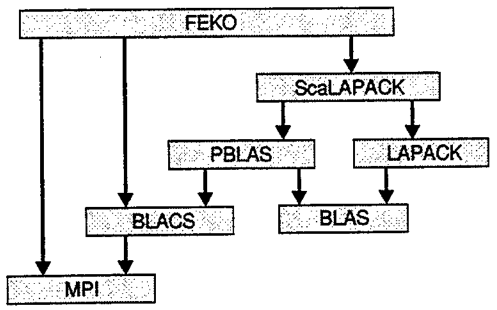


Fig. 6: Call graph of the main program FEKO and the libraries.

Quite often we are also interested in a solution for multiple right hand sides $\vec{y}$ (loop indicated by the dashed arrow in Fig. 1), therefore an LU factorization with subsequent back-substitution is preferred here.

One possible candidate for the solution of the system of linear equations might be PETSc [36], but motivated by the performance of ScaLAPACK [37, 38, 39] (see Fig. 5 and Refs. [38, 35] for more performance details) we chose the latter package, which is the parallel and distributed memory version of LAPACK [40]. According to the call graph in Fig. 6, ScaLAPACK requires the BLAS library, its parallel version PBLAS [41] as well as BLACS [42]. Besides the MPI calls in FEKO to the MPI library (e.g. MPICH on our workstation cluster), there are also direct calls to BLACS and ScaLAPACK subroutines. In ScaLAPACK, we use the subroutine PZGETRF to perform the LU factorization of the matrix A, PZGECON to get an estimate of its condition number, and PZGETRS to obtain the solution by back-substitution.

At the end of 1997 an alternative to ScaLAPACK became available. The PLAPACK [43] code claims to outperform ScaLAPACK for larger problem sizes, we are currently investigating this package as well.

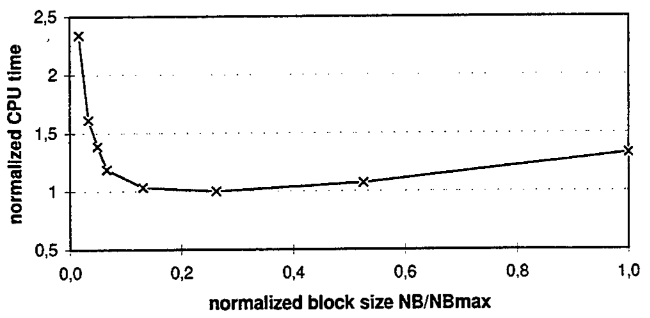


Fig. 7: Typical dependency of the normalized CPU–time required for matrix solve (LU factorization with ScaLAPACK) as a function of the normalized block size $\frac{N_B}{N_{Bmax}}$ with $N_{Bmax} = \lceil \frac{\tilde{N}}{p} \rceil$.

It was already mentioned above and will again be shown below (see Fig. 10) that the block size N_B is a very important parameter for an efficient matrix fill. However, also the CPU–time required for the solution of the system of linear equations depends on N_B (see Fig. 7 for a typical relation we found from experiments).

3.4 Parallel field computation

Normally, when the conventional MoM is applied to electrically large structures, the CPU times for matrix fill and solving the system of linear equations are dominant (proportional to N^2 and N^3, respectively). The time required to compute electric and magnetic near– and far–fields, which is proportional to N and to the number of observation points, remains relatively small in this case.

However, with the hybrid extensions as described in Section 2.2, we are able to reduce N drastically, so that

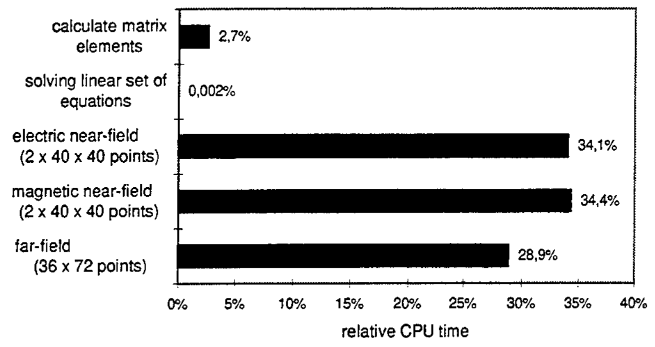


Fig. 8: CPU time requirement for the different phases of the solution process for an analysis of a mobile telephone radiating close to the human head (special Green's function).

the time required to compute the fields might become dominant. This is illustrated in Fig. 8, where the relative CPU–time for the different phases of the solution process is given for an analysis of a mobile telephone radiating at a frequency of 900 MHz close to the human head. The free–space Green's function has been replaced by the Green's function for a layered dielectric sphere [26]. One plane of symmetry was used, so that the $N_1 = 189$, $N_2 = 5$ and $N_3 = 6$ basis functions lead to $\tilde{N} = 99$ unknowns. The near–field was computed inside the head in two planes ($\vec{E}$ is required for the specific absorption rate SAR), and the far–field was calculated on a spherical surface with 36×72 observation points in order to compute the radiated power.

A master/server concept was chosen for the parallelization, which allows an almost perfect dynamic load balancing: The master process distributes the tasks to the remaining $p-1$ server processes, which compute the near– or far–field. As soon as one server process has finished its calculation, it reports the result back to the master process. By this message, the master process is notified that the server process is ready and the master sends the coordinates of another observation point $\vec{r}$ to the server process. By this method, it is guaranteed that the server processes are always busy with numerical computations, i.e. server processes running on faster workstations don't have to wait for other processes running on slower machines.

Some possible drawbacks of this method are:

- The master process has to buffer the results it receives from the server processes before writing them to the output file, because the order how the results are received may be quite arbitrary. In the output file, however, the results shall be printed in the correct order specified by the user, e.g. with increasing x when the field strength is computed along the x–axis.
- From p processes only $p-1$ are actually involved in the calculation. However, the loss of efficiency by a factor of $\frac{p-1}{p}$ is small for large values of p. Also on clusters of connected workstations we usually start the master and one server process on the same workstation, so that p processes are running on $p-1$ workstations and all processors of these workstations are involved in the field calculation.

4 Parallelization of the hybrid extensions

The hybrid extensions by PO and UTD, as briefly summarized in Section 2.2, lead to a drastically reduced size of the matrix. The main changes concerning the parallelization as compared to the conventional MoM are that during the calculation of the matrix elements a_{mn} all the coupling effects between the different regions (MoM/PO/UTD) as indicated in Fig. 3 have to be taken into account. Therefore, with reference to the total solution time, the matrix filling remains relatively time consuming. The time for the matrix solve, however, becomes negligible for most applications.

For the matrix we keep the one–dimensional block–cyclic row distribution scheme, so that an entire row of the matrix resides on a node. After computing the MoM interaction, the modification due to PO or UTD can be performed locally on this node without any communication, since we keep the required geometry information for ray–tracing etc. on each node.

Especially concerning the coupling of MoM with UTD, the time for near– or far–field computations might be dominant for complex geometries with many surfaces and edges, since for every observation point ray–tracing must be performed in order to find possible ray paths with reflections and diffractions between source and observation point. The parallelization of the field computation as described in Section 3.4 can be applied without

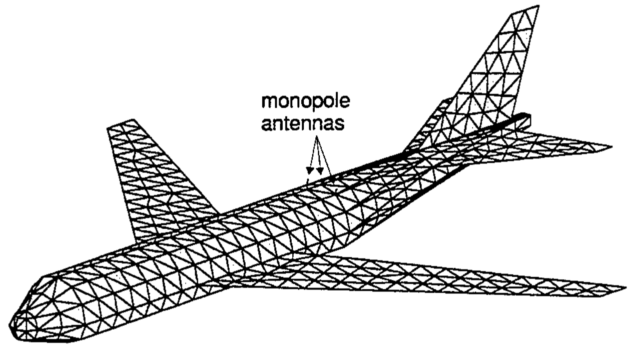


Fig. 9: Model of an aircraft consisting of 1256 triangular patches and 36 wire elements.

any modification to the MoM/UTD hybrid method as well. The implemented dynamic load balancing scheme is very important in this case, since for different locations of the observation point the number of ray paths might vary from zero to several hundreds or even thousands for complex geometries, which is also reflected in large variations of the required CPU–time for the ray–tracing procedure and the subsequent field computation.

5 Examples and parallel performance

For benchmarking purposes, the simple model of an aircraft as shown in Fig. 9 was used. The structure is excited by 3 small monopole antennas on top of the fuselage. The coupling between the antennas was of interest, but the far–field radiation pattern and the near–field were computed as well. The model consists of 1256 triangular patches and 36 wire elements, leading to $N_1 = 1850$ rooftop basis functions $\vec{f}_n^1$ on triangular patches, $N_2 = 33$ basis functions $\vec{f}_n^2$ on wire elements, and $N_3 = 20$ basis functions $\vec{f}_n^3$ at connection points. There is one plane of symmetry, so that the total number of $N = N_1 + N_2 + N_3 = 1903$ unknowns can be reduced to $\tilde{N} = 975$ ($\tilde{N}_1 = 940$, $\tilde{N}_2 = 22$, and $\tilde{N}_3 = 13$).

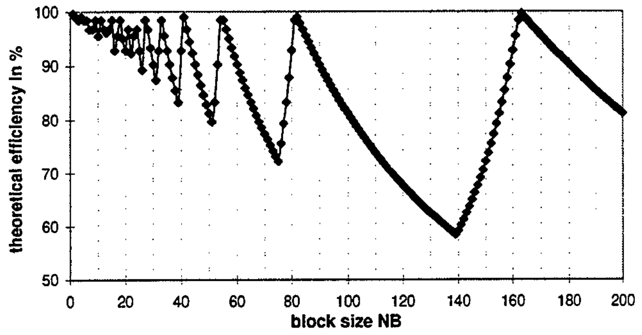


Fig. 10: Theoretical efficiency for the matrix filling process on $p = 6$ nodes with $\tilde{N} = 975$ rows as a function of the block size N_B.

For the first investigation, we used $p = 6$ nodes on an Intel Paragon. The matrix with $\tilde{N} = 975$ rows and $N = 1903$ columns is distributed among the PEs according to Fig. 4 with a block size N_B in the range $1 \leq N_B \leq \lceil \frac{\tilde{N}}{p} \rceil = 163$. If $N_B = 163$ is selected, the first 5 nodes have 163 rows of the matrix each, while the last process stores only the remaining 160 rows. This means that during matrix fill the last process has to wait $3T$ where T denotes the time to compute a matrix row (here we assume that the time T is the same for all the matrix rows). The theoretical efficiency based on the fact that some processes have to wait for others to finish the computations is then

$$\frac{5 \cdot 163 \cdot T + 1 \cdot 160 \cdot T}{6 \cdot 163 \cdot T} = \frac{975}{6 \cdot 163} = 99.69\,\%$$

(communication times etc. have been neglected). However, a block size of for instance $N_B = 140$ results in 7 blocks (6 blocks with 140 rows on nodes $1 \ldots 6$ and 1 last block with 135 rows on node 1). The theoretical efficiency is then much lower, namely

$$\frac{1 \cdot 275 \cdot T + 5 \cdot 140 \cdot T}{6 \cdot 275 \cdot T} = \frac{975}{6 \cdot 275} = 59.09\,\%.$$

The dependency of this theoretical efficiency on the block size N_B is plotted in Fig. 10. For our example here, useful block sizes are $163, 82, 55, 41, 33, 27, \ldots$. From this set the final choice is made automatically so that we expect the matrix solve to be most efficient (based on experiences on the different target machines, see Fig. 7). For the following investigations the maximum block size $N_B = 163$ is selected.

Fig. 11 shows the time required to compute the matrix elements on each of the 6 nodes. As already mentioned in Section 3.2, the CPU time required to compute a single matrix element a_{mn} differs over a wide range, since some of the integrals contain singularities whilst others are evaluated numerically with adaptive integration

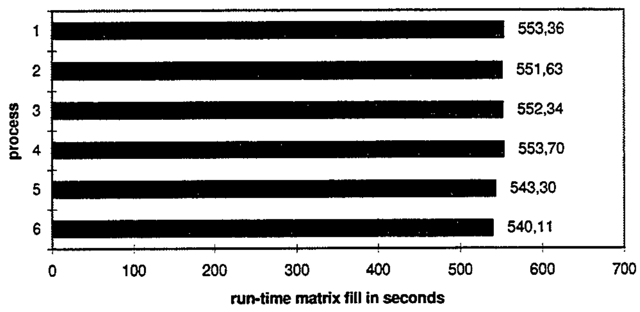


Fig. 11: Run–time for the matrix fill on the different nodes for $p = 6$ processes (without using the efficient fill technique), $\tilde{N} = 975$, $N = 1903$.

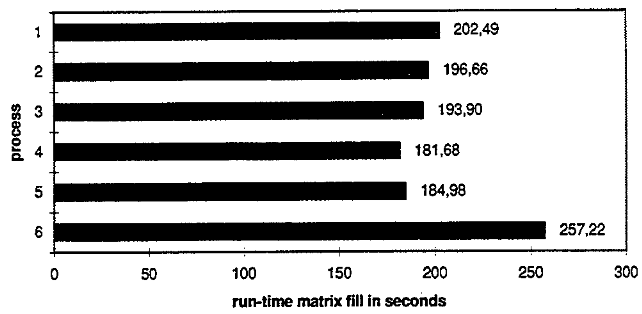


Fig. 12: Run–time for matrix fill on the different nodes for $p = 6$ processes (using the efficient fill technique but no mapping function), $\tilde{N} = 975$, $N = 1903$.

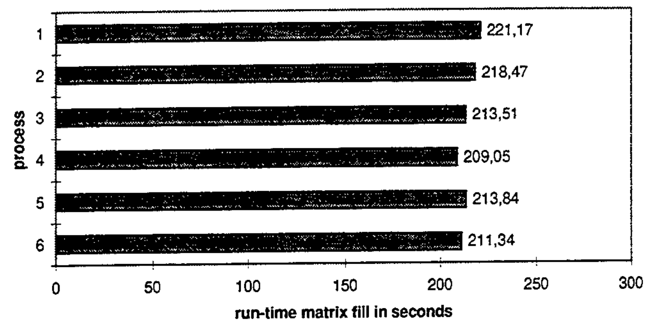


Fig. 13: Run–time for the matrix fill on the different nodes for $p = 6$ processes (using the efficient fill technique and a special mapping function to interchange matrix rows), $\tilde{N} = 975$, $N = 1903$.

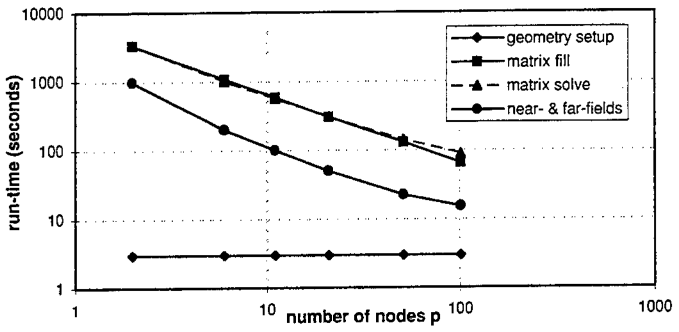


Fig. 14: Run–time of the various phases of the parallelized MoM solution for the example of the plane depicted in Fig. 9 as a function of the number of processes (no usage of symmetry and no efficient matrix fill), $N = 1903$.

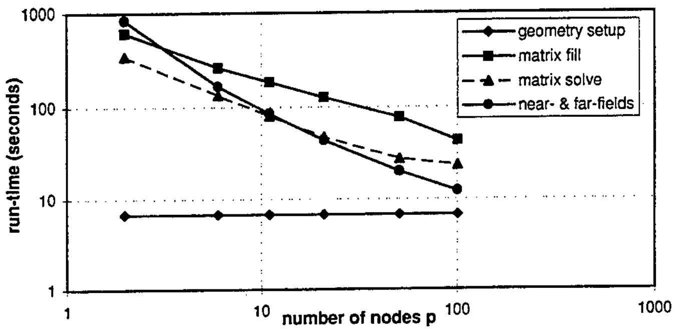


Fig. 15: Run–time of the various phases of the parallelized MoM solution for the example of the plane depicted in Fig. 9 as a function of the number of processes (usage of symmetry and efficient matrix fill), $\tilde{N} = 975$, $N = 1903$.

schemes. But it can be seen from Fig. 11 that on average all the 6 processes take about the same time to compute the matrix elements. The difference is only in the range of a few percent.

If, however, the efficient matrix fill technique is used, the situation changes as shown in Fig. 12. Of course the run times are reduced. The factor on the nodes 1 to 5 is 2.86 on average, which is quite satisfactory when compared to the reduction factor of 3.07 for the sequential version (using no symmetry, the reduction factor is usually somewhat higher, e.g. for the sequential version it is 3.32 for this particular example). The high ratio of $\frac{2.86}{3.07} \approx 0.93$ indicates that in most cases all the matrix elements to which the computed integrals (loop over triangular patches) have a contribution are kept on the same node. If we increase the number of processors, then this ratio will decrease. But even for $p = 100$ nodes (block size only $N_B = 10$), the ratio is still 1.75 on average, the maximum for one node is 2.62.

From Fig. 12 it can be seen that with the efficient fill technique the last process 6 takes about 30 % longer than the other nodes. The reason for this is that apart from the last 125 rows of the sub-matrices A_{1j} ($j = 1 \ldots 3$, see Fig. 2), the six sub-matrices A_{2j} and A_{3j} ($j = 1 \ldots 3$) with 22 and 13 rows, respectively, are kept entirely on this node and it takes longer to compute these elements.

The mapping function discussed in Section 3.2 can be used to overcome this problem. We also distribute the sub-matrices A_{2j} and A_{3j} to the different nodes, so that for instance for our example considered here with $N_B = 163$ the first node contains 157 consecutive rows of the sub-matrices A_{1j}, 4 rows of A_{2j} and 2 rows of A_{3j}. The resulting CPU–time requirement on the different nodes is depicted in Fig. 13. As opposed to Fig. 12 without mapping function, the total CPU time for the matrix fill can be decreased from 257.2 sec to 221.2 sec.

Another investigation is depicted in Figs. 14 and 15. We have executed FEKO again on the Intel Paragon, but now using a variable number p of nodes. The run–time of the different phases of the solution process according to the flow–chart in Fig. 1 has been measured and is plotted in Fig. 14 (symmetry was not used and no efficient matrix fill) and in Fig. 15 (usage of symmetry and efficient matrix fill) as a function of p. As explained in Section 3.1, the geometrical setup is performed sequentially by one process, i.e. the run–time for this part is constant. The other run–times show a satisfactory decrease with increasing p. The ideal case would be a decrement by a factor of 10 with 10 times more nodes, i.e. a linear diagonal line in the double logarithmic diagrams.

The example of the plane is relatively small to be considered a real HPC problem. However, we have used this problem to optimize the code and perform a large number of tests. Some data for larger problems are also available. For instance, for EMC purposes we analyzed the surface currents and field distribution inside a car when a plane wave is exciting the structure with $N = 15864$ basis functions. The model has no symmetry, therefore also

Table 1: Wall clock times for an EMC investigation of a car on a CRAY T3E with $N = \tilde{N} = 15864$ unknowns.

	64 nodes	128 nodes	ratio
geometry setup	8.12 sec	7.56 sec	1.074
matrix fill	390.23 sec	195.91 sec	1.992
matrix solve	1174.48 sec	616.86 sec	1.904
near- & far-fields	39.85 sec	20.66 sec	1.929
total solution time	1616.14 sec	844.77 sec	1.913

$\tilde{N} = 15864$. The resulting wall clock times are tabulated in Table 1. The last column gives the ratio of the times for the runs on 64 and 128 nodes, respectively. A value of 2 is expected for perfect scaling, our values show a very promising scaling factor (except for geometry setup which has not been parallelized).

For near- and far-field calculations, the performance of the dynamic load balancing scheme, i.e. the automatic adaptation to the different CPU speeds and to the actual load of the workstations, shall also be demonstrated. The problem used to illustrate the scheme is the analysis of a mobile communications antenna radiating close to the user's head. The hybrid MoM/Green's function technique as briefly discussed in Section 2.2 is applied. For this kind of formulation the CPU-time for the field computation is dominant, see Fig. 8. But also for some of the hybrid formulations, e.g. when the time consuming ray-tracing of the UTD must be performed, the CPU-times for near- and far-field computations and matrix fill dominate over the matrix solve time.

FEKO was executed on a heterogeneous cluster of 15 different workstations with different speeds (see Fig. 16 a) for relative speed) during office hours, so that there were also other users logged in (see Fig. 16 b) for the number) and there were also some background processes from other users running on some of the machines (see Fig. 16 c)).

Fig. 16 d) shows the relative number of computations carried out by the different server processes running on the 15 workstations. Even though there were no other background jobs on the last 4 workstations no. 12 to 15, they calculated the near- and far-field for only about 1.5% of the observation points each. The reason for this can be seen from Fig. 16 a): These 4 workstations are rather slow when compared to the other ones, so that the master process automatically takes this speed difference into account. Another example to demonstrate the dynamic load balance: If one compares the first two workstations of the same model and same speed, one can see that the process 1 computed only about half the number of field points when compared to process 2. Again Fig. 16 c) can be used to explain this behavior: There was one background process by another user on workstation 1.

Results for the speedup on a homogeneous cluster of identical workstations (no. 1 to 10 in Fig. 16 a)), which were exclusively available to the FEKO job during the test, i.e. there were no other users present and no back-

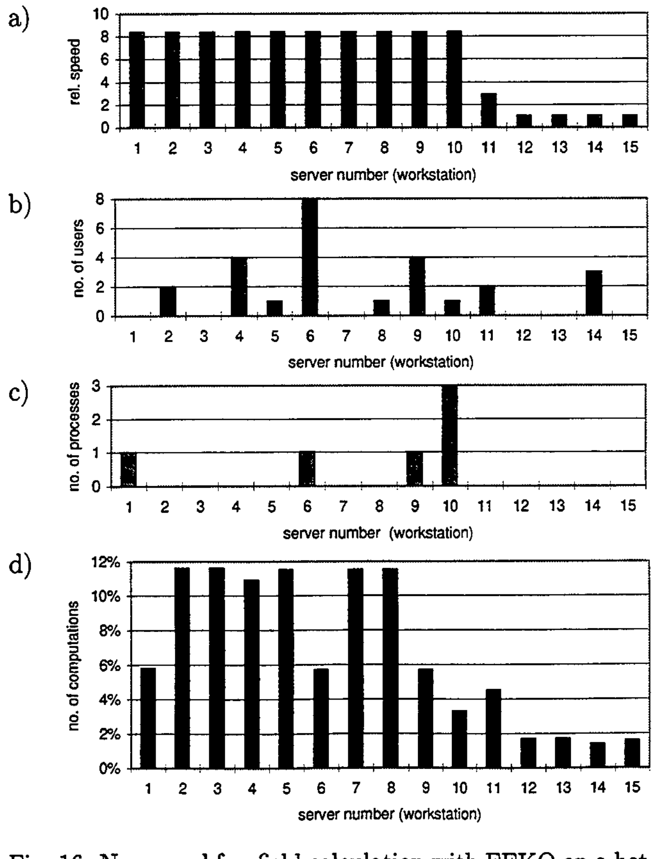


Fig. 16: Near- and far-field calculation with FEKO on a heterogeneous cluster of 15 different workstations during office hours with other users present in the network.

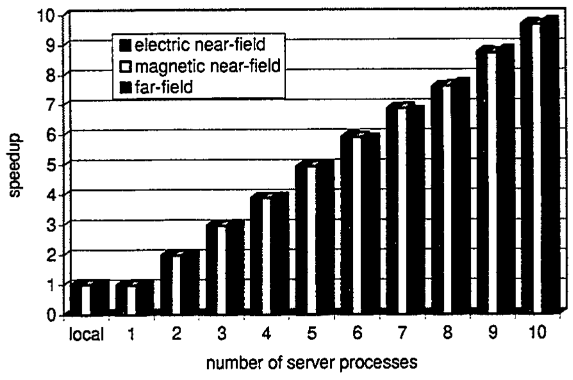


Fig. 17: Speedup for the near- and far-field calculation with a special Green's function on a homogeneous cluster of 10 IBM RS 6000 (Model 43P) workstations.

Table 2: Speedup for the near– and far–field calculation with a special Green's function on the Intel Paragon.

no. of server nodes	10	20	60	90	110
total no. of nodes p	11	21	61	91	111
speedup for calc. of					
electric near–field	9.99	19.52	59.14	87.78	96.70
magn. near–field	9.98	19.50	59.09	88.12	96.86
far–field	9.95	19.00	56.36	85.10	99.98

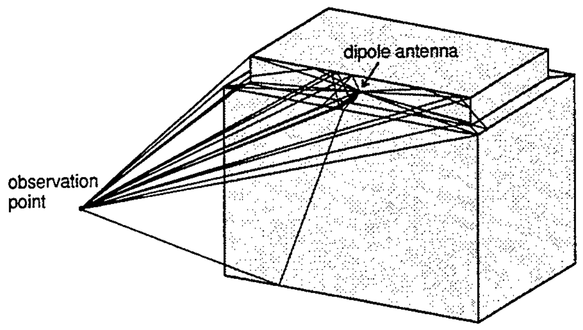


Fig. 18: Dipole antenna radiating on top of a building. Some ray paths of the UTD are shown for an observation point in the near–field.

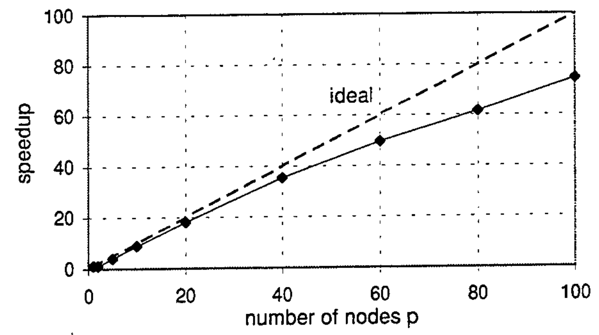


Fig. 19: Speedup for the calculation of the radiation pattern of the dipole antenna on top of a building (hybrid MoM/UTD method executed on the Intel Paragon).

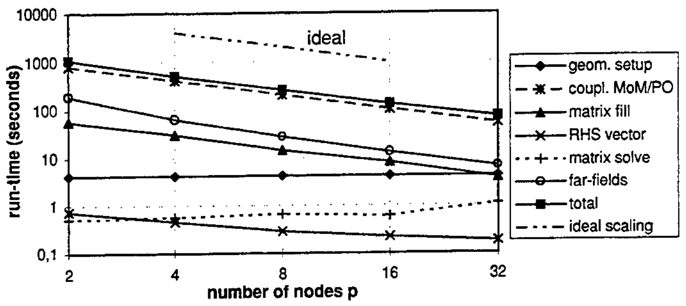


Fig. 20: Run–time on the CRAY T3E for the various phases of the MoM/PO hybrid method for the analysis of a mobile communications antenna on the roof of a car (10059 basis functions in total).

ground jobs were running, are given in Fig. 17. As mentioned above, the number of server processes corresponds to the number of workstations that were used, the master process was run together with one of the server processes on the same workstation.

Table 2 lists the achieved speedup on the Intel Paragon for the same computation.

An example for the application of the MoM/UTD hybrid method is depicted in Fig. 18, where a dipole antenna is radiating at a frequency of 900 MHz on top of a building. The building is modeled by 14 flat polygonal plates, and for the UTD we consider direct, reflected, edge and corner diffracted rays. Some of these rays are also shown in Fig. 18 for an observation point in the near–field. However for determining the performance of the parallel implementation, we calculated the far–field radiation pattern in 3 planes for a total number of 2880 observation directions.

The achieved speedup on the Intel Paragon is plotted in Fig. 19. For $p = 100$ nodes the speedup is about 75.

One final example shall be considered demonstrating the application of the MoM/PO hybrid method. A mobile communications antenna on the roof of a car was analyzed resulting in $N = \tilde{N} = 54$ basis functions in the MoM–region and 10005 basis functions in the PO–region. Full coupling between these regions (see Fig. 3) is taken into account, so that the computation of the coupling coefficients is the most time consuming part of the solution process, see dashed line in Fig. 20. These coupling coefficients are required for the computation of the matrix elements and the RHS vector. It can again be seen from Fig. 20 that the geometry setup has not been parallelized, but also here for the matrix solve process the scaling is very poor. However, the time for this phase is almost negligable, and we cannot expect a better scaling for only 54 MoM unknowns. The total solution time (solid line with square symbols) shows an excellent scaling behavior, almost parallel to the ideal curve.

6 Validation of the results

The results of the FEKO code have been extensively validated during the past years by comparison with measurements or with other published reference data. We have also performed numerous calculations by comparing the different methods available in FEKO, e.g. a mobile telephone radiating in front of a spherical head model can be treated by the conventional MoM, the hybrid MoM/PO method and also by the special Green's function technique.

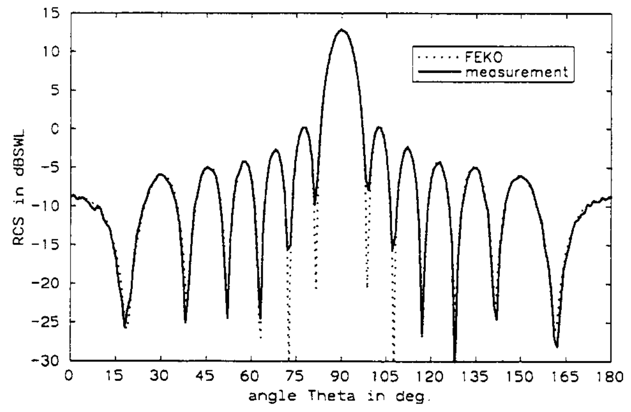


Fig. 21: Computed and measured monostatic radar cross section of a brass strip with dimensions 63.3 × 6.3 × 0.32 mm^3 at 15 GHz.

As a validation example, Fig. 21 shows the predicted monostatic RCS of a brass strip with dimensions 63.3 × 6.3 × 0.32 mm^3 at a frequency of 15 GHz. The measured results have been published by the David Florida Laboratory, Ottawa, Canada [44].

7 Conclusions

A parallel implementation of the MoM and hybrid code FEKO in a distributed memory environment has been presented. The parallelization is based on the MPI standard, so that all major massively parallel supercomputers and also clusters of connected workstations are supported. Promising performance results were presented for an Intel Paragon, a CRAY T3E and a cluster of workstations of 10 identical IBM RS 6000 Model 43P and 5 slower IBM RS 6000 workstations. Especially on the heterogeneous cluster of 15 workstations we were able to demonstrate the efficiency of the dynamic load balancing scheme for the near- and far-field computations, which might dominate the total solution time for some hybrid techniques.

Current investigations concentrate on out-of-core solutions and also on improving the performance. We are evaluating PLAPACK and iterative techniques in PIM with preconditioners as an alternative to the LU factorisation in ScaLAPACK.

References

[1] R. F. Harrington, *Field Computation by Moment Methods.* Macmillan Company, New York, 1968.

[2] T. Cwik, "Parallel decomposition methods for the solution of electromagnetic scattering problems," *Electromagnetics*, vol. 12, pp. 343–357, 1992.

[3] V. F. Fusco, "Electromagnetic field modelling using parallel and distributed processing techniques," in *IEE 3rd International Conference on Computation in Electromagnetics, Bath*, pp. 13-16, Apr. 1996.

[4] Y. S. Choi-Grogan, K. Eswar, P. Sadayappan, and R. Lee, "Sequential and parallel implementations of the partitioning finite-element method," *IEEE Trans. on Ant. and Prop.*, vol. 44, pp. 1609-1616, Dec. 1996.

[5] T. Cwik, D. S. Katz, and J. Patterson, "Scalable solutions to integral-equation and finite-element simulations," *IEEE Trans. on Ant. and Prop.*, vol. 45, pp. 544–555, Mar. 1997.

[6] Y. Lu and C. Y. Shen, "A domain decomposition finite-difference method for parallel numerical implementation of time-dependent Maxwell's equations," *IEEE Trans. on Ant. and Prop.*, vol. 45, pp. 556-562, Mar. 1997.

[7] W. A. Imbriale and T. Cwik, "A simple physical optics algorithm perfect for parallel computing architecture," in *10th Annual Review of Progress in Applied Computational Electromagnetics, Monterey*, pp. 434–441, Mar. 1994.

[8] D. M. Elking, J. M. Roedder, D. D. Car, and S. D. Alspach, "A review of high-frequency radar cross section analysis capabilities at McDonell Douglas Aerospace," *IEEE Ant. and Prop. Magazine*, vol. 37, pp. 33–43, Oct. 1995.

[9] T. Cwik, J. Partee, and J. Patterson, "Method of moment solutions to scattering problems in a parallel processing environment," *IEEE Trans. on Magnetics*, vol. 27, pp. 3837–3840, Sept. 1991.

[10] D. I. Kaklamani and A. Marsh, "Solution of electrically large planar scattering problems using parallel computed method of moments technique," *Journal of Electromagnetic Waves and Applications*, vol. 9, pp. 1313–1337, Oct. 1995.

[11] D. I. Kaklamani and A. Marsh, "Benchmarking high-performance computing platforms in analyzing electrically large planar conducting structures via a parallel computed method of moments technique," *Radio Science*, vol. 31, pp. 1281–1290, Sept. 1996.

[12] D. I. Kaklamani, K. S. Nikita, and A. Marsh, "Extension of method of moments for electrically large structures based on parallel computations," *IEEE Trans. on Ant. and Prop.*, vol. 45, pp. 566–568, Mar. 1997.

[13] G. Mäder and F. H. Uhlmann, "A parallel implementation of a 3D-BEM-algorithm using distributed memory algorithms," *IEEE Trans. on Magnetics*, vol. 33, pp. 1796–1799, Mar. 1997.

[14] J. M. Putnam and J. D. Kotulski, "Parallel CARLOS-3D code development," in *12th Annual Review of Progress in Applied Computational Electromagnetics, Monterey*, pp. 228–235, Mar. 1996.

[15] S. P. Walker and C. Y. Leung, "Parallel computation of time-domain integral equation analyses of electromagnetic scattering and RCS," *IEEE Trans. on Ant. and Prop.*, vol. 45, pp. 614–619, Apr. 1997.

[16] "MPI: a message passing interface standard, version 1.1," tech. rep., University of Tennessee, Knoxville, June 1995. http://www.mcs.anl.gov/mpi/index.html.

[17] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI — Portable Parallel Programming with the Message-Passing Interface.* The MIT Press, Cambridge, 1994.

[18] W. Gropp, E. Lusk, A. Skjellum, and N. Doss, "A high–performance, portable implementation of the MPI message interface standard," tech. rep., Argonne National Laboratory, University of Chicago, 1996. URL http://www.mcs.anl.gov/mpi/mpich.

[19] S. M. Rao, D. R. Wilton, and A. W. Glisson, "Electromagnetic scattering by surfaces of arbitrary shape," *IEEE Trans. on Ant. and Prop.*, vol. 30, pp. 409–418, May 1982.

[20] U. Jakobus and F. M. Landstorfer, "Novel basis function for the equivalent magnetic current in the method of moments solution of dielectric scattering problems," *Electronics Letters*, vol. 29, pp. 1272–1273, July 1993.

[21] U. Jakobus and F. M. Landstorfer, "Improved PO–MM hybrid formulation for scattering from three-dimensional perfectly conducting bodies of arbitrary shape," *IEEE Trans. on Ant. and Prop.*, vol. 43, pp. 162–169, Feb. 1995.

[22] U. Jakobus and F. M. Landstorfer, "Improvement of the PO–MoM hybrid method by accounting for effects of perfectly conducting wedges," *IEEE Trans. on Ant. and Prop.*, vol. 43, pp. 1123–1129, Oct. 1995.

[23] U. Jakobus and F. M. Landstorfer, "Current–based hybrid moment method analysis of electromagnetic radiation and scattering problems," *ACES Journal*, vol. 10, pp. 38–46, Nov. 1995.

[24] U. Jakobus and F. M. Landstorfer, "Hybrid MM–PO–Fock analysis of monopole antennas mounted on curved convex bodies," in *12th Annual Review of Progress in Applied Computational Electromagnetics, Monterey*, pp. 101–108, Mar. 1996.

[25] U. Jakobus and F. M. Landstorfer, "A combination of current– and ray–based techniques for the efficient analysis of electrically large scattering problems," in *13th Annual Review of Progress in Applied Computational Electromagnetics, Monterey*, pp. 748–755, Mar. 1997.

[26] H.-O. Ruoß, U. Jakobus, and F. M. Landstorfer, "Efficient EM analysis of hand–held mobile telephones close to human head using modified method of moments," *Electronics Letters*, vol. 31, pp. 947–948, June 1995.

[27] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes, *Computer Graphics: Principles and Practice.* Addison–Wesley, Reading, 1993.

[28] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing: Design and Analysis of Algorithms.* The Benjamin/Cummings Publishing Company, Inc., Redwood City, 1994.

[29] D. B. Davidson, "Parallel matrix solvers for moment method codes for MIMD computers," *ACES Journal*, vol. 8, pp. 144–175, July 1993.

[30] D. B. Davidson, "Large parallel processing revisited: A second tutorial," *IEEE Ant. and Prop. Magazine*, vol. 34, no. 5, pp. 9–21, 1992.

[31] R. D. da Cunha and T. Hopkins, *PIM 2.2: The Parallel Iterative Methods package for Systems of Linear Equations User's Guide (Fortran 77 version)*, 1997. URL http://www.mat.ufrgs.br/pim-e.html.

[32] J. Shen, T. Hybler, and A. Kost, "Preconditioned iterative solvers for complex and unsymmetric systems of equations resulting from the hybrid FE–BE method," *IEEE Trans. on Magnetics*, vol. 33, pp. 1764–1767, Mar. 1997.

[33] G. Bürger, H.-D. Brüns, and H. Singer, "Advanced method of moments based on iterative equation system solver," in *IEEE 1997 International Symposium on Electromagnetic Compatibility, Austin, Texas*, pp. 236–241, Aug. 1997.

[34] J. Rahola, "Iterative solution of dense linear systems in electromagnetic scattering calculations," in *14th Annual Review of Progress in Computational Electromagnetics, Monterey*, vol. II, pp. 1126–1133, Mar. 1998.

[35] L. S. Blackford and R. C. Whaley, "ScaLAPACK evaluation and performance at the DoD MSRCs," tech. rep., Department of Computer Science, University of Tennessee, Knoxville, 1998.

[36] S. Balay, W. Gropp, L. C. McInnes, and B. Smith, *PETSc 2.0 Users Manual*, 1995. URL http://www.mcs.anl.gov/petsc/petsc.html.

[37] J. Choi, J. J. Dongarra, L. S. Ostrouchov, A. P. Petitet, D. W. Walker, and R. C. Whaley, "The design and implementation of the ScaLAPACK LU, QR, and Cholesky factorization routines," tech. rep., Oak Ridge National Laboratory, Sept. 1994.

[38] J. Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, "LAPACK working note 95: ScaLAPACK: A portable linear algebra library for distributed memory computers — design issues and performance," tech. rep., University of Tennessee, 1995.

[39] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, *ScaLAPACK User's Guide*, 1997. URL http://www.netlib.org/scalapack/scalapack-home.html.

[40] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK Users' Guide — Release 2.0*, Sept. 1994. URL http://www.netlib.org/lapack/index.html.

[41] J. Choi, J. Dongarra, S. Ostrouchov, A. Petitet, D. Walker, and R. C. Whaley, "LAPACK working note 100: A proposal for a set of parallel basic linear algebra subprograms," tech. rep., University of Tennessee, May 1995.

[42] J. J. Dongarra and R. C. Whaley, "LAPACK working note 94: A user's guide to the BLACS v1.0," tech. rep., University of Tennessee, June 1997.

[43] R. A. van de Geijn, *Using PLAPACK: Parallel Linear Algebra Package.* The MIT Press, Cambridge, 1997.

[44] C. L. Larose, C. W. Trueman, and S. R. Mishra, "Measured RCS polar contour maps for code validation," *ACES Journal*, vol. 11, pp. 25–43, Nov. 1996.

Running SuperNEC on the 22 Processor IBM-SP2 at Southampton University

D C. Nitch, A P.C. Fourie and J S. Reeve

Jeff Reeve
Department of Electronics and Computer Science
University of Southampton
Southampton
SO17 1BJ, UK

Derek Nitch / Andre Fourie
Department of Electrical Engineering
University of the Witwatersrand
South Africa
email:nitch@odie.ee.wits.ac.za
email:fourie@odie.ee.wits.ac.za

ABSTRACT

SuperNEC (SNEC) is an object-oriented version of NEC-2 which has been modified to execute on a network of distributed memory processors. The matrix filling, solving and pattern computation routines are capable of running in parallel. A number of structures have been simulated using this code on the 22 processor IBM-SP2 machine at Southampton University. The principal problem studied was the DC-3 at 90 MHz. LU decomposition and an iterative matrix solution scheme were used in the study. The simulation time for this structure (which includes 3-D radiation patterns) dropped from 2.5 hours on a single processor to about 17 minutes when simulated on 12 processors using LU decomposition. Execution times are about half of these times when using the iterative solver. The far field patterns obtained from the simulation are compared with measured data and show good agreement. The largest problem tackled on the IBM machine was the DC-3 simulated at 160 MHz. This problem requires 17035 segments and was simulated in 5.3 hours on 21 processors.

INTRODUCTION

SuperNEC is an object-oriented electromagnetics code that implements the same theory as the FORTRAN program NEC2 [1]. The program is considerably more advanced that NEC2, as it incorporates features such as model based parameter estimation (MBPE), fast iterative solvers and a UTD-MOM hybrid capability amongst other features. One of its primary advantages is that SNEC has been written to operate in parallel on a network of processors.

The advantage of being able to operate in parallel is two fold. First, there is a potential reduction in the time taken to solve a problem. Secondly, a network of processors generally has more memory at its disposal than a single processor, thus much larger problems may be solved on a network of processors.

This paper reports on the performance of SNEC on the distributed memory, parallel IBM-SP2 machine located at Southampton University. The principal problem analysed is a DC-3 at 90 MHz. This structure is modelled using 5500 segments and generates an interaction matrix of 240 Mbytes. Run times for larger problems are also presented.

A Brief History of SNEC

In 1989 the FORTRAN program, NEC2, was modified to operate in parallel on a network of transputers [2-3]. During this project, all the numerically intensive routines were re-written to operate in parallel. The efficiencies achieved for the filling and factoring of the interaction matrix approached 90 percent, and even higher efficiencies were achieved for radiation pattern and near field calculations. As such, the runtime performance of the code was very satisfactory, however, the effort required to change the FORTRAN code was considerable. This was not necessarily due to the structure of the code, but rather due to the paradigm in which it was written. The use of common blocks (global variables) and implicit variable typing were two of the main features of the program which hindered the development and debugging of the parallel code. Making further modifications to the FORTRAN code was therefore predicted to be as difficult as adapting the code to run in parallel. One method of bypassing this problem was to rewrite NEC2 using a different programming paradigm [4]. The paradigm chosen for the implementation was the object-oriented paradigm, and the language of implementation, C++.

Converting FORTRAN to C++ is not merely a matter of translating from one language to the other. Rather, it requires the identification of the physical and mathematical entities present in the problem domain (segments, two-port networks, matrices etc.) and the assignment of responsibilities and attributes to each of these entities. For example, one of the elements identified in NEC2 was the concept of a wire segment. A segment was assigned the responsibilities of being able to move and rotate itself. The attributes assigned to a segment include the starting and end co-ordinates of the wire and the radius of the segment.

The primary aim of the object-oriented design was to make changing the electromagnetic parts of the code as easy as possible. For example, one obvious requirement is the ability to change the basis functions used in the MOM solution. This was partly achieved by introducing the concept of a Propagator class. The responsibilities of a Propagator are primarily to describe how an element in the structure propagates. That is, given some current distribution (known only to the Propagator and the Current class), a Propagator is able to compute the electric and magnetic fields at any point in space. Encapsulating the basis function in the Propagator class allows the entire code to be written independent of the function used to represent the current in a segment of the structure. Changing the basis function in SNEC requires only the writing of a new Propagator class and supplying the associated boundary conditions. Thus it is considerably easier to change the basis function in SNEC than in its FORTRAN counterpart.

The object-oriented version of NEC was completed in 1993. To illustrate the ease with which the new program could be modified, the sequential, object-oriented program was adapted to run on a network of transputers. This task took a total of two weeks to complete.

Transputer technology has severe limitations, the most overbearing being that a transputer machine is not an all-purpose machine and hence is not widely available (when compared to workstations). Thus restricting SNEC to a network of transputers was not a good long term prospect for the parallel code.

Currently there are a number of communications libraries that allow processors to communicate with one another over a local area network. One of these libraries is PVM (Parallel Virtual Machine). This library allows one to build up a parallel machine using a heterogeneous collection of existing processors. For example, one could connect a number of SUN 10's and RISC 6000 machines to form a fairly powerful parallel machine.

SNEC was freed from the limitations of the transputer by using PVM to implement the required communications. The parallel routines used in the PVM implementation are very similar to those used in the transputer version [2]. Subsequently, SNEC has been adapted to incorporate a hybrid MOM-UTD method, MBPE, fast iterative solvers and many other features.

The IBM SP2 machine at Southampton University.

The POWER parallel System (SP2) at Southampton University is a 22 processor distributed memory parallel machine. The first 16 nodes are called 'thin1' nodes, and have the following configuration :

Processor	66 MHz Power2
Memory	128 Mbytes
Memory Bus	64 bit
Instruction Cache	32 kbit
Data Cache	64 kbit
Disks	Two 2Gbyte SCSI disks

Table 1 : Specifications of nodes 1-16

Nodes 17 to 22 are 'thin2' nodes with the following configuration :

Processor	66 MHz Power2
Memory	256 Mbytes
Memory Bus	128 bit
Instruction Cache	32 kbit
Data Cache	64 kbit
Disks	Two 2Gbyte SCSI disks

Table 2 : Specifications of nodes 17-22

There is a single log-on node which is used primarily for testing, debugging and submitting jobs to the machine.

All processors are connected by "IBM high performance S2 adapters". This communications switch generates a maximum theoretical point-to-point bandwidth of 40 Mbits/s.

Communication libraries available on the SP2 include MPI (Message passing interface) and PVM.

Generating the SNEC model of the DC-3

SNEC requires that the structure to be modelled be specified in terms of straight wire segments. Generating such a model for the DC-3 is an exceptionally tedious task when tackled by hand. One of the tools developed at the University of the Witwatersrand for the conversion of complicated structures into wire segments is a package called SIG (Structure Interpolation and Gridding package) [7]. The input to SIG are curves that define the cross section of the structure at various points along its length. The cross-sections are defined in an ASCII file in a specified format. Given this information and the frequency at which the structure is to be modelled, SIG produces a valid wire element model suitable for input to either NEC2 or SNEC. The user defined cross-sections and the model generated by SIG for the DC-3 at 90 MHz are given in Figure 1 and Figure 2 respectively.

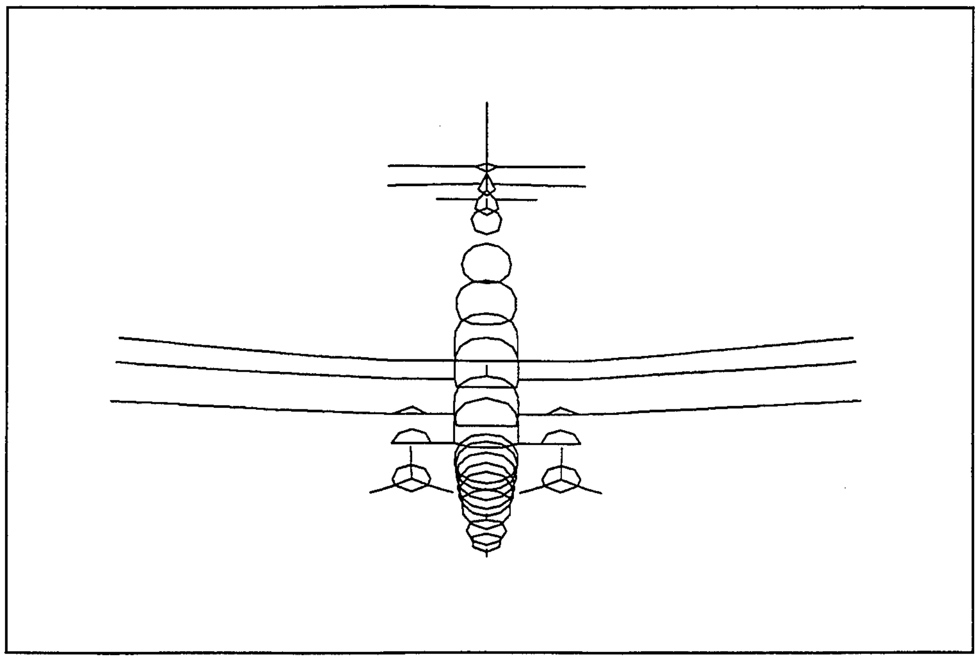


Figure 1 : The user defined cross-sections for the DC-3.

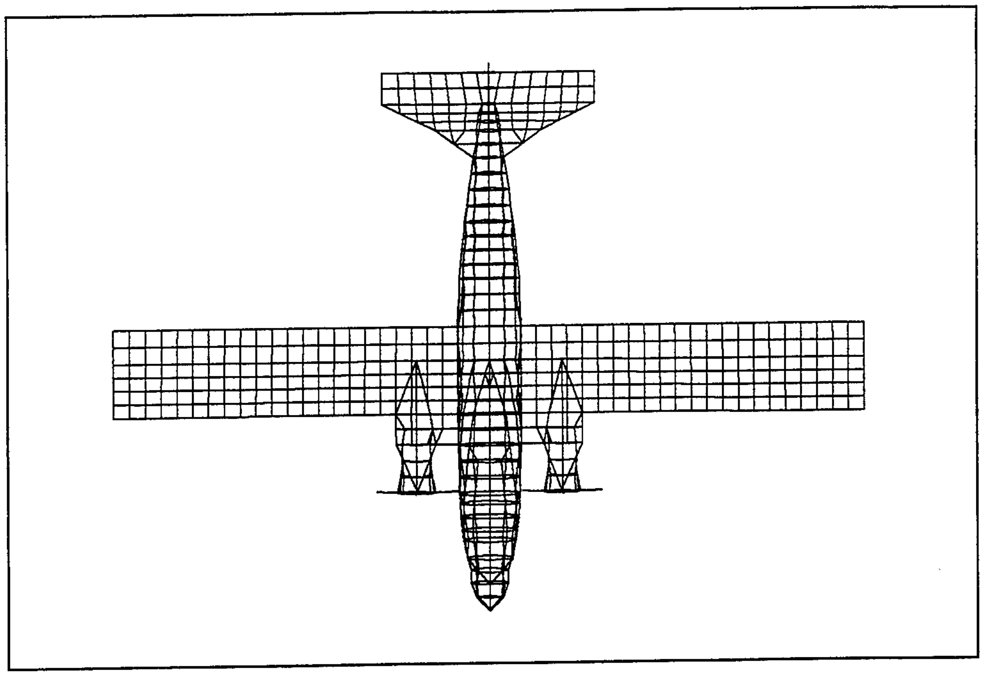


Figure 2 : The top view of the SIG generated model of the DC-3.

Results

The results presented in this section discuss the performance of SNEC on the SP2 and compare the far field patterns obtained from the simulation with measured values. These results will be discussed separately in the following sections.

Performance on the SP2

The simulation was performed on a network of 1 to 12 processors. The performance of the parallel code is assessed in terms of its speed up and execution times. The speed up of a parallel program is defined as :

$$\boxed{\text{speedup} = \frac{\text{simulation time on one processor}}{\text{simulation time on p processors}}}$$

The first stage in the MoM procedure is that of filling the interaction matrix. When executed in parallel, each processor is allocated an equal portion of the matrix to fill. Since filling the matrix does not require any communication between processors, it is expected that the speed up increases linearly with an increase in the number of processors. In actual fact, the speed up increases slightly faster than linear. The reason for this super-linear speedup could be attributed to the fact that when many processors are used, each processor has less memory to manipulate and this results in a more efficient use of cache memory.

The second stage of the MoM procedure is solving the matrix equation, for which SNEC has a number of different methods. In this study, the parallel performance of two of these techniques are investigated. The first technique involves factoring the matrix and then performing backward and forward substitution. This method is a parallel implementation of the matrix solving routines used in NEC2. The second technique is an iterative solver. The iterative solver is a two stage process that uses the Sparse Iterative Method (SIM) [5] for the initial iterations, and then, when the rate of convergence of this solver falls below a pre-set threshold, the bi-conjugate gradient stabilised algorithm is used to complete the iterative process. The SIM is similar to the banded matrix iteration (BMI) used in the GEMACS code. A comparison of the two methods is given in [8], whist the rationale behind the combination of the stationary and non-stationary techniques is detailed in [5].

Performance graphs for each major stage of the MoM solution have been generated.

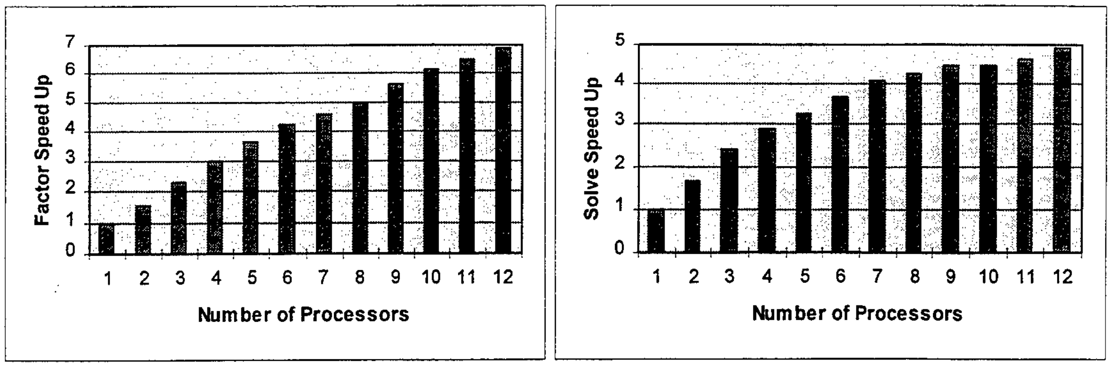


Figure 3 : The speed up of the LU Solver.

Figure 3 shows the speed up of the factoring and solving stages of the LU solver. The slow increase in speed up of this algorithm is attributed to the fact that there is not enough computation for each processor to amortise the communication overhead. Increasing the size of the problem on a 12 processor network will result in a more efficient use of the available processing power. The improvement in speed up could not be demonstrated in this study as the 5500 segment (240Mbyte) problem is the largest problem that could be simulated on a single processor with the memory available.

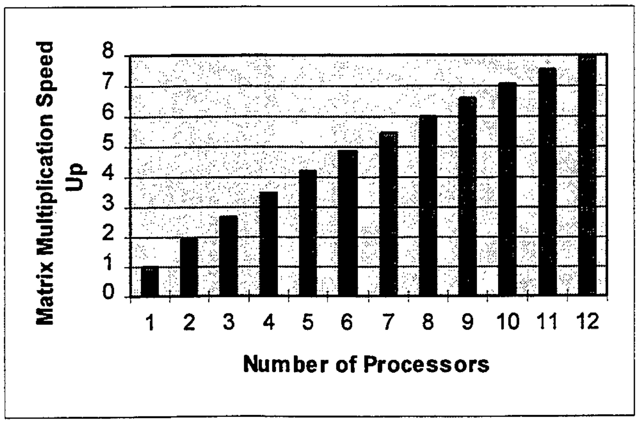


Figure 4 : Speed up attained by the matrix multiplication computation.

Most of the time used in the iterative solver is spent during a matrix-vector product operation. The speed up of the parallel matrix-vector product is given in Figure 4.

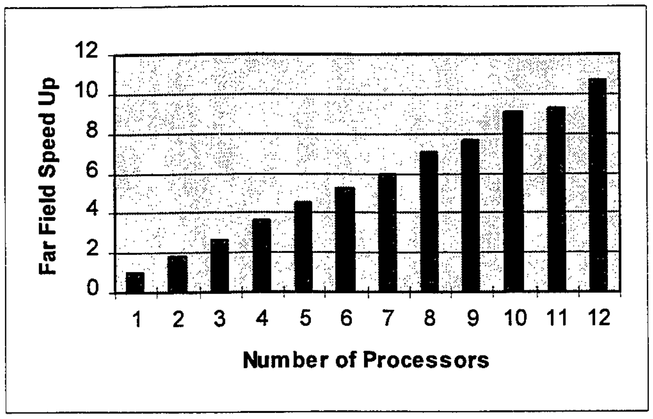


Figure 5 : The speed up of the far field computation.

In the simulation of the DC-3, four radiation patterns were computed. Three patterns were one-dimensional cuts taken in 2° increments. The fourth pattern was a three-dimension radiation pattern sampled every 4°. The speed up of this computation is fairly consistent with the increase in the number of processor in the network and is displayed graphically in Figure 5.

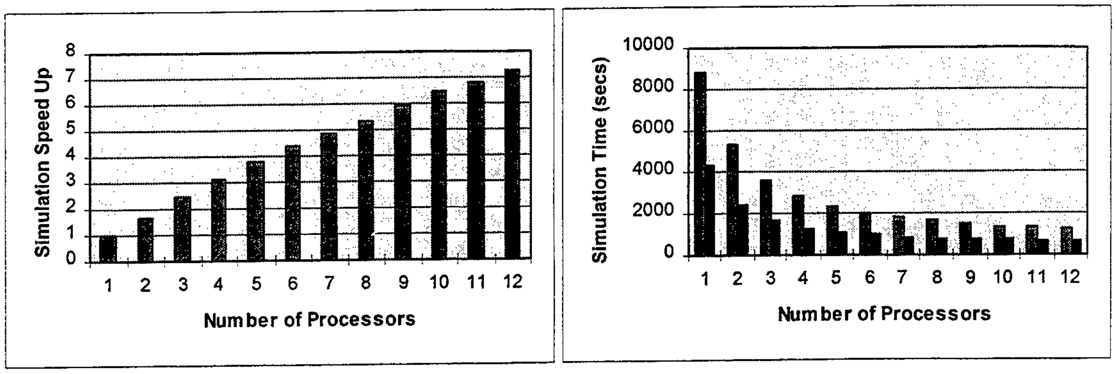


Figure 6 : The speed up of the simulation and run times on the SP2.

The first graph in Figure 6 shows the speed up of the entire simulation when using the LU solver. The execution times for the simulation are given in the second graph of this figure. The light-coloured bars in the second graph correspond to the run-times when using the LU solver, whilst the dark bars depict the run-times for the iterative solver. The iterative solver was deemed to have converged when the average tangential electric field error was less than 10^{-8}.

The DC-3 has been simulated on the IBM-SP2 machine at higher frequencies, the performance of the SP2 is depicted in Table 3.

Frequency (MHz)	Number of Segments	Number of Processors	Memory (Gbytes)	Memory per Processor (Mbytes)	Simulation Time (hours)
148	14608	16	1.7	106	3.7
160	17035	21	2.3	110	5.3

Table 3 : Run times for larger problems on the SP2 using the LU solver.

Comparison with measured data

The following figures show the theoretical and measured far fields for the DC-3 in the azimuth, pitch and roll planes. The measured patterns were performed using a 1 in 72 scale model and were obtained by Fourie, Givati, Clark and Pascoa [6].

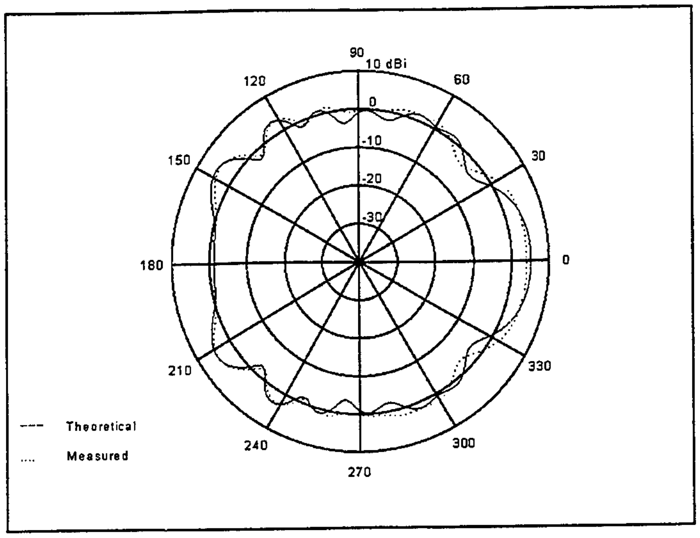


Figure 7 : The azimuth-plane radiation pattern of the top fin antenna at 90 MHz.

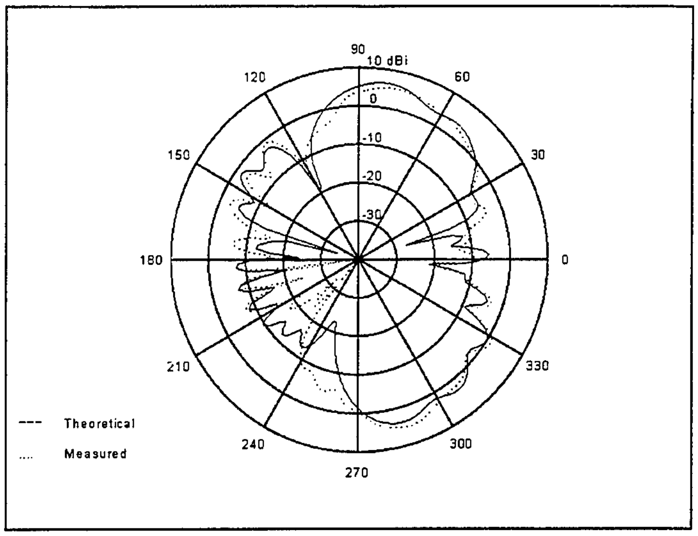


Figure 8 : The pitch roll radiation pattern of the top fin antenna at 90 MHz.

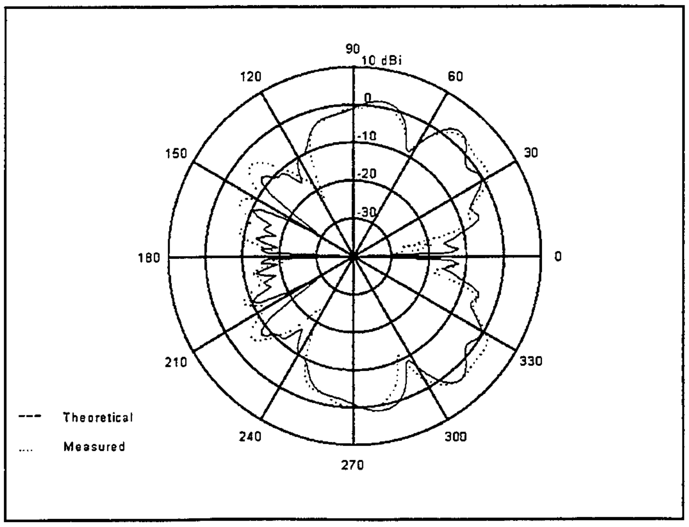


Figure 9 : The side roll radiation pattern of the top fin antenna at 90 MHz.

Conclusion

The parallel performance of SNEC on the IBM-SP2 distributed memory machine has been presented. The speed up attained by SNEC varied between 1.8 for a two processor network to 7.3 for a 12 processor network. The speed up of the larger networks will approach the number of processors for larger problems.

The largest problem reported is the simulation of the DC-3 at 160 MHz. Solving this problem on a sequential machine would require 2.3Gbytes of memory (RAM or hard-disk) and take about 4 days of computer time to simulate. Distributing the problem onto 21 processors, reduces the memory per processor requirement to 110 Mbytes and a solution was found in 5.3 hours. This demonstrates the effective use of distributed memory and processing power.

This study was done on a specialised parallel processing machine, however the software is not limited to execution on such machines. Workstations, linked by a local area network maybe used to form a reasonably powerful parallel processing machine.

References

[1]Burke G.J., Poggio A.J., "Numerical Electromagnetics Code (NEC2) - Method of

Moments," Naval Oceans Systems Center, San Diego, CA Tech Doc 116, 1981.
[2] D. C. Nitch and A. P. C. Fourie. "Adapting the numerical electromagnetics code to run in parallel on a network of transputers." Applied Computational Electromagnetics Society Journal, 5(2):76--86, 1990.
[3] D. C. Nitch and A. P. C. Fourie. "Parallel Implementation of the Numerical Electromagnetics Code", Applied Computational Electromagnetics Society Journal, Vol. 9, No. 1, March 1994, pp 51-57.
[4] D. C. Nitch and A. P. C. Fourie. "A Redesign of NEC2 Using the Object Oriented Paradigm", IEEE Antennas and Propagation Society International Symposium, Vol. 2, Seattle, June 1994, pp 1150-1153.
[5] D. C. Nitch and A. P. C. Fourie. "A Sparse Iterative Method (SIM) for Method of Moments Calculations", IEEE Antennas and Propagation Society International Symposium, Vol. 2, Seattle, June 1994, pp 1146-1149.
[6] A. P. C. Fourie, O. Givati, A.R. Clark and N. Pascoa. “Measured and Computer Results of Air-to-Ground communication performance of 2 aircraft at VHF/UHF frequencies”, ANTEM-96 Symposium, Quebec, Canada, August 1996.
[7] A. P. C. Fourie, D. C. Nitch and O. Givati, “A Complex-Body Structure Interpolation and Gridding Program (SIG) for NEC”, IEEE Antennas and Propagation Magazine, Vol. 36, No. 3 June 1994, pp 85-59.
[8] A. P. C. Fourie, D. C, Nitch “Comparing the Sparse Iterative Method (SIM) with the Banded Jacobi and Conjugate Gradient Techniques”, IEEE Antennas and Propagation Society International Symposium, Vol. 2, Seattle, June 1994, pp 1181-1184.

Highly Parallel Implementation of the 3D Integral Equation Asymptotic Phase Method for Electromagnetic Scattering[1]

Xianneng Shen, *RS/6000 System Division, IBM Corporation*
Aaron W. Davis, *Computer Science Department, University of Washington*
Keith R. Aberegg, *ERIM International, Ann Arbor, MI*
Andrew F. Peterson, *School of Electrical & Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332-0250, peterson@ee.gatech.edu*

ABSTRACT: In this paper, we discuss the implementation of the 3D Integral Equation–Asymptotic Phase (IE–AP) method using the parallel architecture IBM RS/6000 SP. The IE-AP method is a hybrid numerical/asymptotic approach for electromagnetic scattering that attempts to reduce the number of unknowns required to accurately model electrically large structures. The IE-AP method will be described, and results will be reported for the parallel matrix fill implementation, and the relative performance of the PESSL and PETSc toolkits for parallel matrix solution.

1. Introduction

The application of interest is the radar cross-section (RCS) computation for a 3D arbitrarily shaped electrically large target. Practical RCS prediction using numerical methods has long been thought of as unrealistic because of the excessive computation and memory requirements. The RCS prediction of a fighter-sized aircraft, for example, requires the solution of a matrix equation whose dimension can easily exceed 10^6. The cost of such computations also discouraged efforts to improve other aspects of computational electromagnetics. The successful development of massively parallel processing (MPP) technologies improves the opportunity to solve these problems [1-4].

In this paper, we discuss the development and implementation of the 3D Integral Equation – Asymptotic Phase (IE-AP) code using a Massively Parallel Processing (MPP) architecture. The IE-AP method is a hybrid numerical/asymptotic approach for electromagnetic scattering that incorporates an asymptotic phase function in an attempt to reduce the number of required unknowns by as much as an order of magnitude. The formulation was originally implemented as a traditional serial process [5-6]. In order to fully test the IE-AP approach, the method was implemented on a parallel-architecture IBM RS/6000 SP. An overview of the IE-AP formulation is presented in the following section, while the remainder of this article is focused on the parallel implementation using the IBM SP and some of the observations we made during this process.

2. The IE-AP Formulation

The Integral Equation – Asymptotic Phase (IE–AP) method is a hybrid numerical/asymptotic procedure for determining the current density and far fields associated with electromagnetic scatterers. The method attempts to eliminate the traditional dependence on a 10-per-wavelength unknown density, without limiting the shape of the target under consideration or the accuracy. The method has been developed and tested on 2D [5] and 3D [6] perfectly conducting scatterers, and the initial results suggest an order of magnitude reduction in the necessary unknown density. In the present implementation of the method, the current density is taken to be the product of an asymptotic result (the phase of the physical optics current) and a residual vector function that is solved for using the method of moments. The motivation for the approach is the idea that the traditional 10-per-wavelength unknown density is primarily dictated by the phase progression of the electromagnetic wave or current (360 degrees per wavelength), and therefore over large portions

[1] This work was supported by the Cornell Theory Center, by a Supercomputing Program for Undergraduate Research Grant from the National Science Foundation, and by NSF ECS-9257927.

of targets an asymptotic result can provide the necessary phase progression. By treating the unknown currents as a product of the asymptotic result and an unknown residual function, it is thought that the residual function will be slowly varying and therefore can be represented by far fewer unknowns per wavelength.

The two-dimensional implementation of the IE–AP method reported in [5] involved the combined-field equation (CFIE), curved (parabolic) cells, subsectional quadratic polynomial basis functions, Dirac delta (TM) or pulse (TE) testing functions, and the incorporation of the appropriate TM edge singularity at corners. Test geometries considered in [5] consisted of circular cylinders, elliptic cylinders, wedge-cylinder structures, and square cylinders, all of which had perimeter dimensions in the range of 44 to 71 wavelengths. In most cases, the IE–AP result obtained with an average unknown density of one per wavelength or less differed by no more than 3 percent from a reference numerical solution obtained with the traditional 10-unknowns-per-wavelength density. In the 2D case, the method appears to produce an order-of-magnitude reduction in unknown density without significant additional error in the results. Based on the apparent success of the 2D approach, the idea was extended to three dimensions [6].

To eliminate interior resonance difficulties that arise with the analysis of electrically large scatterers with the electric-field or magnetic-field equations [9-10], the 3D CFIE was employed. The CFIE for scattering from perfectly conducting targets can be written in the form

$$\alpha \bar{E}^{inc}_{tan} + (1-\alpha)\eta\, \hat{n} \times \bar{H}^{inc} = -\,\alpha \left(\frac{\nabla\nabla\cdot\bar{A} + k^2\bar{A}}{j\omega\varepsilon} \right)_{tan} + (1-\alpha)\eta\, (\bar{J}_s - \hat{n} \times \nabla \times \bar{A}) \quad (1)$$

where α is an arbitrary parameter in the range $0 < \alpha < 1$, η is the medium impedance, ε is the permittivity, ω is the radian frequency,

$$\bar{A}(u,v) = \iint \bar{J}_s(u',v')\, \frac{e^{-jkR}}{4\pi R}\, du'dv' \quad (2)$$

and R is the distance from the source point (u′,v′) to the observation point (u,v), which in (1) is located an infinitesimal distance outside the scatterer.

Assuming that the incident field is a uniform plane wave, the current in Equation (1) is replaced by the expansion

$$\bar{J}_s(u,v) \cong e^{jk(x\sin\theta\cos\phi + y\sin\theta\sin\phi + z\cos\theta)} \sum_n j_n\, \bar{B}_n(u,v) \quad (3)$$

where (x,y,z) denotes a point on the surface of the scatterer with a local parametric representation (u,v), and (θ,ϕ) denotes the spherical angle from which the incident wave propagates. The exponential function represents the asymptotic phase associated with the uniform plane wave excitation; equivalently this is the phase arising from the physical optics approximation of the surface current. In contrast to the classical physical optics approximation, however, we use (3) throughout the lit and shadow portions of the target.

Our implementation uses triangular patches with parabolic curvature to represent the scatterer, and a type of razor-blade testing function to enforce the equation [8]. Based on our experience with the 2D results, it was decided to implement higher-order basis functions for the 3D case instead of the traditional Rao-Wilton-Glisson (RWG) functions, which provide at most a linear representation. The vector basis functions we implemented have one higher polynomial order in each variable than the RWG functions, and provide a linear representation of the normal vector component and a quadratic representation of the tangential vector component (LN/QT) along cell edges prior to mapping [7, 10]. The mapping process used to incorporate patch curvature is described in [8]. Reference [7] provided data showing the improved accuracy of the LN/QT functions on curved patches compared with the RWG functions on flat facets, and demonstrated a reduction of 2-3 in the required number of unknowns. [Note that this improvement does not take into account the incorporation of the asymptotic phase function in Equation (3).]

A few preliminary results were generated for the 3D IE–AP method, incorporating the asymptotic phase function, using spheres and cone-spheres as test cases [6]. Spheres permit a comparison with the exact solution, while cone-spheres permit comparison with alternative formulations that incorporate axial symmetry (so-called "body of revolution" codes). For example, Figures 1a, 1b, and 1c show the surface currents and scattering cross section for a sphere of radius 3λ. These results were obtained using 720 unknowns, for an average unknown density of about 6.5 unknowns per square wavelength of surface area. Note that there are discontinuities in the current, more evident in the ϕ component, which are typical of any expansion in divergence-conforming vector basis functions (such as the RWG functions and the LN/QT functions used here). For the particular mesh used to produce these data, the scattering cross section exhibits some error in the 60-90 degree sector.

As a second example, consider a cone-sphere with a total length of 10 wavelengths, an endcap radius of 1.48 λ, and a cone half angle of 7.5 degrees. Figure 2 shows the scattering cross section obtained with the IE-AP formulation using 1600 unknowns, for a total average density of 29 unknowns per square wavelength. A reference solution obtained using a "body of revolution" code with 40 unknowns per wavelength along the generating arc is shown for comparison. Although the agreement is acceptable over most of the range, the data show some apparent error in the 0-80 degree portion of the plot.

The preceding results suggest that a substantial reduction in unknowns might be possible with the IE-AP approach, as compared with a traditional subsectional integral equation formulation that almost always requires more than 100 unknowns per square λ. However, the specific results in Figures 1 and 2 contain error, and better scatterer

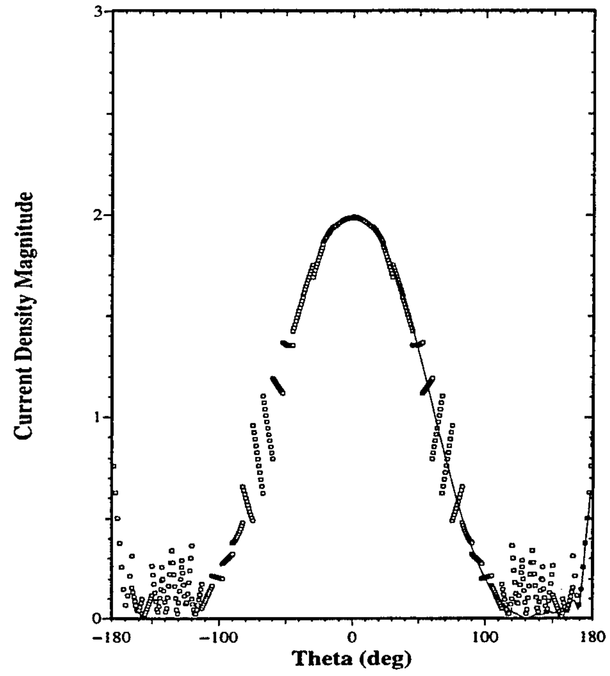


Figure 1b. The ϕ-component of the surface current density induced on a sphere of radius 3 λ. The CFIE result (markers) is compared with the exact solution (solid curve).

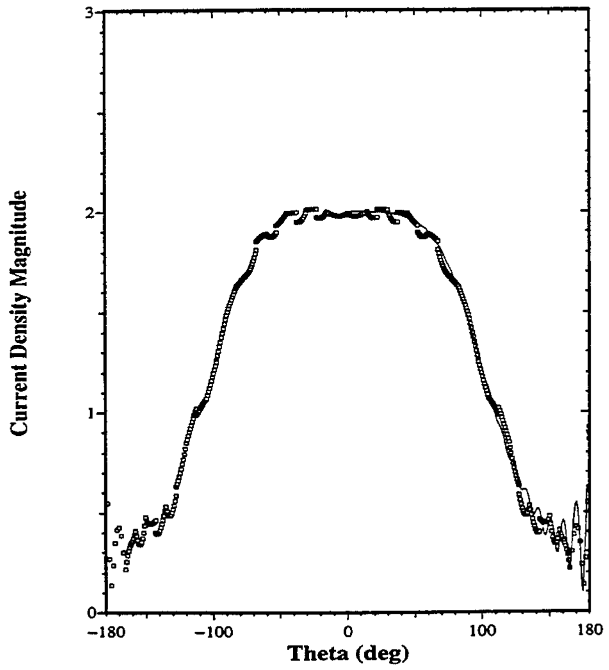


Figure 1a. The θ-component of the surface current density induced on a sphere of radius 3 λ. The CFIE result (markers) obtained using the IE-AP expansion with only 720 unknowns is compared with the exact solution (solid curve).

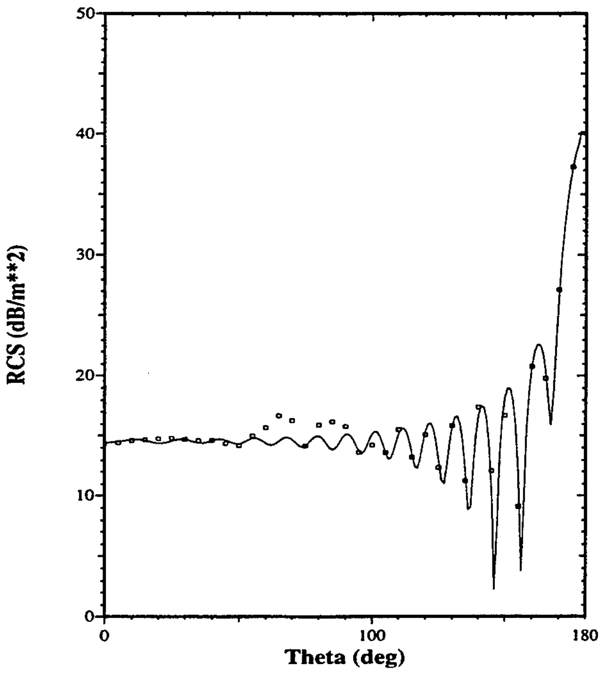


Figure 1c. The scattering cross section for a sphere of radius 3 λ. The CFIE result obtained using the IE-AP expansion with only 720 unknowns (markers) is compared with the exact solution (solid curve).

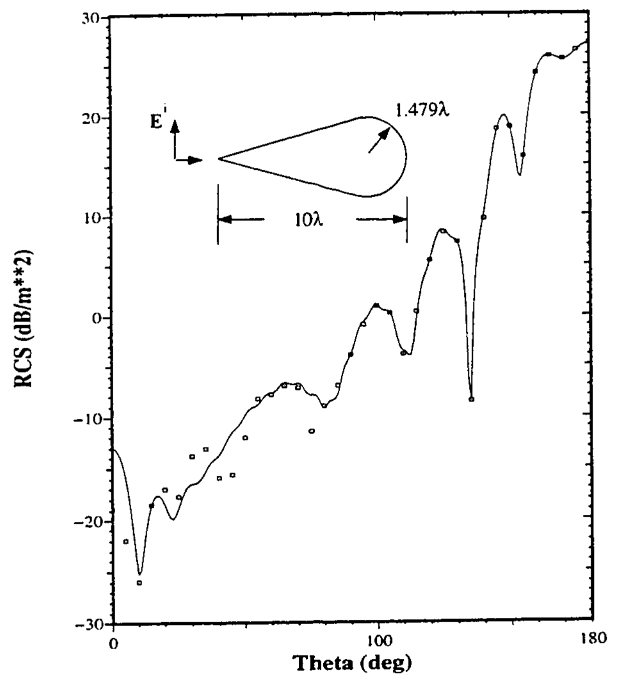


Figure 2. The scattering cross section of a cone-sphere of length 10 λ, an endcap radius of 1.48 λ, and a cone half angle of 7.5 degrees. Results obtained using the IE-AP formulation with 1600 unknowns (markers), a density of 29 unknowns per λ^2, are compared with a reference solution obtained using a "body of revolution" code with 40 unknowns per wavelength along the generating arc (solid curve).

meshes are necessary to improve the accuracy of the results. These test cases were meshed empirically, with many of the cells electrically large (less than 10 unknowns per wavelength over most of the cone-sphere, but with a much higher density of cells near the tip). Since the cell sizes arising within the IE-AP implementation can range from the usual size (0.1 λ) to cells that can be quite large (perhaps 10 λ in some situations) it is difficult to arrive at a nearly optimal scatterer model without feedback from the numerical solution. We feel that some form of adaptive gridding is necessary to optimize the cell dimensions based on local error estimates obtained from a coarse-cell model. In addition, one would expect that the savings from the IE-AP procedure would become greater as the electrical size of the scatterer increased. Unfortunately, available computer resources limited the range of scatterers easily analyzed in [6] to those of about 10 wavelengths in linear dimension. In order to treat larger problems and study the IE–AP method in detail, the present authors teamed with the support of the Cornell Theory Center to adapt the method to the parallel-architecture IBM SP machine.

Because of the reduced number of unknowns, and the additional cost associated with the entries of the matrix equation, the IE–AP method represents a shift of the traditional computational burden away from the matrix solve part of the process and toward the matrix fill part. Consequently, the procedure might be easier to parallelize with a high efficiency than a traditional integral equation formulation. In the following sections, we consider the parallelization of the 3D IE-AP algorithm for the IBM SP architecture.

3. Parallel Implementation

There are several distinct parallel-architecture computer systems available [1-2, 11]. They can be cataloged as shared memory, distributed shared memory, and distributed memory systems. We implemented the parallel IE–AP code on the Cornell Theory Center's 512-node IBM RS6000 SP system using the Message-Passing Interface (MPI). The IBM RS6000 SP employs a distributed memory architecture with relatively powerful nodes and a fast interconnect network, the SP switch network. The SP supports three classes of parallel programming models: message-passing, data parallel, and shared memory. An explicit message-passing program can run very efficiently since it matches the underlying SP system architecture. The message-passing library can be accessed from a FORTRAN program.

The explicit message-passing approach using MPI has a number of advantages. MPI is a standard message-passing library, and most parallel computer vendors have implemented MPI to facilitate code portability despite underlying hardware differences.

Because the IE–AP matrix entries involve a traditional Green's function multiplied with an asymptotic phase function, and integral domains that may span several wavelengths, the matrix entry computation is somewhat more costly than with a traditional 10-unknown-per-wavelength integral equation code. For N unknowns, the computational complexity of the matrix fill procedure is N^2 while the matrix factorization involves complexity of order N^3. The coefficient of the N^2 term in the complexity calculation for the IE–AP approach is somewhat larger than in traditional formulations, while the order N is smaller (by perhaps an order of magnitude). Since the matrix fill task represents the majority of the computation for a moderate-sized problem, the parallelization study initially focused on that part of the procedure. Subsequently, a parallel LU factorization was implemented using two available libraries.

The simple implementation of the matrix fill is discussed in the following section. The remainder of the paper is devoted to a discussion of the parallel LU factorization procedure.

4. Parallel Matrix Fill

The matrix entries arising from integral equation formulations are usually completely independent expressions, and therefore the matrix fill task is easy to parallelize with high efficiency [2-4, 12-14]. In our initial implementation, every processor or node performs the identical initialization tasks prior to the matrix fill. Assuming P+1 available processors (or nodes) with one used as a dedicated processor, the task of computing each matrix entry is partitioned into P + 1 subtasks and distributed among P processors and dedicated processor. Each subtask involves computing a block of columns (or rows) consisting of either int(N/P) or int(N/P)+1 columns (or rows) of the matrix. When each node finishes computing it sends its block to the dedicated processor, which assembles the global matrix. Thus, each processor does only its share of the computation. In the initial implementation, the dedicated node performed the matrix factorization and solved the matrix equation. There is no communication between nodes during matrix fill since all the computation is carried out locally. Each node only needs to communicate one block of data to the dedicated processor.

If we define speedup as S=(sequential execution time)/(parallel execution time), we are able to achieve almost linear speedup in the matrix fill portion of the program. It is obvious that the limit of the speedup for a P-processor system is P, a ideal speedup. In this case, there is very little overhead in partitioning the data and no message passing is needed during the matrix fill. This is a very simple implementation but it achieves excellent speedup for system with a few thousand unknowns.

Figure 3 shows the speedup achieved for a 1000 unknown system by this approach verses the ideal speedup. There are three speedup curves in Figure 3, labeled *matrix fill*, *optimal*, and *wall clock*. The matrix fill speedup is the ratio of the execution time of the matrix fill in one node to that obtained multi-node. The wall clock time speedup is the ratio of wall clock time used for the entire program execution (including matrix solve) by one node to that required multi-node. The optimal speedup is the ideal case

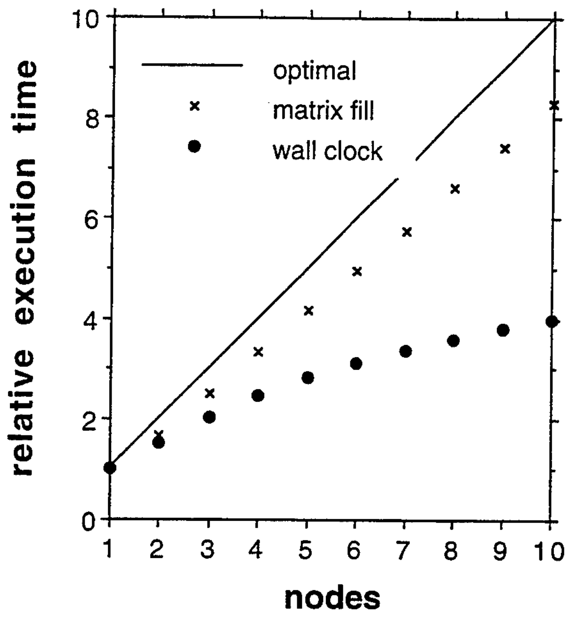


Figure 3. Speedup obtained for a system of order 1000 when the matrix fill task is distributed among various numbers of processors and the matrix solve task is performed serially. The speedup for the matrix fill task and the overall execution (wall clock) are compared with perfect parallelism (optimal).

which achieves 100% parallelism. One can see that the ideal speedup is almost achieved for the matrix fill but poor speedup is observed overall (wall clock speedup). The poor overall performance appears to be the result of two major factors. Let T_{wp} denote the wall clock time of the parallel code execution, and T_{ws} denote the wall clock time of the sequential code execution. The wall clock time speedup S_w is given by $S_w=T_{ws}/T_{wp}$. There are three factors contributed to T_{wp}, the parallel fill execution time, the sequential execution time of the rest of the program (in this case, primarily the matrix factorization), and the communication time between the dedicated node and the other nodes in the application. To improve the wall clock speedup, the matrix solution procedure and the data communication must be optimized.

5. Parallel Matrix Solve

The parallel implementation of the IE-AP matrix fill process using MPI yielded substantial improvement over the serial program. However, it is obviously not efficient to communicate the entire matrix to a single processor for factorization. Although the time to fill the matrix dominates matrix solve time for small problems, this is not the case for systems of order greater than 5000. An additional limiting factor is incurred by communicating the entire matrix to a single processor, which limits the matrix order to a system small enough to fit within the memory of a single processor. The Cornell Theory Center SP thin nodes with 128MB of RAM limit the matrix order to 2400. An out-of-core solution could alleviate this problem, but since we are already computing the matrix elements on many different processors, an easier solution is to eliminate the communication requirement by performing the matrix factorization in parallel.

A large number of articles have proposed and evaluated parallel matrix factorization algorithms [1-2, 11-17]. Since this is a relatively mature area for certain architectures, and libraries for specific machines are in widespread use [17], we chose to investigate parallel numerical libraries specifically recommended for the SP-2. The available libraries were the Portable Extensible Toolkit for Scientific computation (PETSc) and IBM's Parallel Engineering and Scientific Subroutine Library (PESSL). Below, we report on our experience in using these libraries to solve the system of equations in parallel.

5.1 Parallelization Using PETSc

The first attempt to add a parallel matrix solve was made using the PETSc Version 2 Beta 13. PETSc is a parallel numerical library developed at Argonne National Laboratory that attempts to provide a high-level object-oriented interface that automatically manages such details as message passing.

In order to integrate PETSc into our application, we compute the values to be inserted into the PETSc data objects as before. We then call routines telling PETSc where in the global structure to insert the data. Since the programmer is generally unable to directly index individual objects, PETSc maintains a certain level of abstraction, allowing one to change the underlying representation of a data object without actually modifying the program.

After all the data objects have been set up, the application invokes a PETSc solver, which for linear systems is the SLES linear equation solver. In PETSc it is necessary to create a solver context, which may be thought of as an object representing the system. After matrices and vectors have been associated with the system, one calls a generic solve routine to actually solve the system. In its most basic form this process would be done in FORTRAN as follows:

```
call SLESCreate(MPI_COMM_WORLD, sles, ierr)
call SLESSetOperators(sles, A, A,
        DIFFERENT_NONZERO_PATTERN, ierr)
call SLESSetFromOptions(sles, ierr)
call SLESSolve(sles, b, x, iterations,
        ierr)
call SLESDestroy(sles, ierr)
```

This approach has the advantage of being general. By only altering command-line options, it is possible to change the method used to solve the system. The developer can trivially test the utility of different techniques for solving a given system.

After adapting to the peculiarities of the library, we were able to create a working version of our application using PETSc. PETSc allowed us to utilize the routines from the MPI version with little modification, since PETSc uses a slab data partitioning scheme compatible with our original parallel fill routines. This made it possible to call the original fill routines, then easily insert the entries into a PETSc matrix. We then used the generic PETSc solve functions, changing command-line options to experiment with various methods.

We soon discovered that PETSc had severe limitations for the IE–AP type of application, which produces a dense matrix. The PETSc library is targeted at applications with sparse matrices requiring iterative solution. There is no parallel dense LU solver, which forced us to choose between using dense matrices and an iterative solver or using sparse matrices and a block Jacobi solver with LU factorization on individual processor blocks. After some experimentation we chose the latter, although even it did not provide acceptable timing results.

5.2 Results Using PETSc

The PETSc toolkit is not well-suited for our application, as we soon discovered upon viewing timing results. Figure 4 shows the speedup achieved by this approach, which is poor in terms of overall wall clock performance and for the matrix fill part of the computation. The application was executed with the command-line options

```
-mat_mpidense
-pc_type bjacobi
-sub_pc_type lu
```

Despite exhibiting substantial parallel speedup, the

PETSc implementation was much slower than a hand-coded LU routine executing on a single processor.

Upon examining these timings and the output given by running the application with the PETSc option

```
-log_summary
```

we determined that the slow matrix fill time is due almost totally to time spent in the matrix assembly routines. Initially, we believed the problem was the result of our attempting to insert values on the wrong processor and, consequently, being forced to wait for PETSc to communicate the data to the proper node. Unfortunately, we tested a column block partitioning scheme and found even worse results, leading us to believe that PETSc was indeed partitioning as we first suspected but was forced to do some communication during assembly. The execution time could most likely be improved by interleaving some communication between the assembly routines.

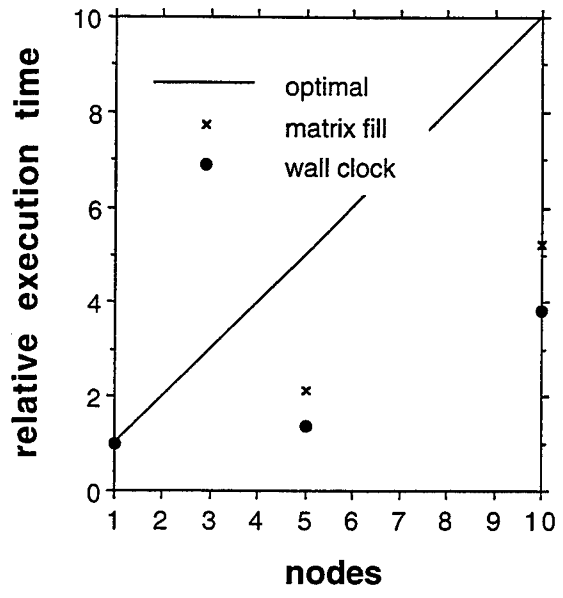


Figure 4. Speedup obtained for a system of order 1000 when the PETSc toolkit is employed. The speedup for the matrix fill task and the overall execution (wall clock) are compared with perfect parallelism (optimal).

The poor times for the solver are easier to explain since it was intended for sparse systems. It is possible that these timing results are not entirely the fault of PETSc, as the library gives the user much control over the algorithms and data types used through the use of command-line options. We may have chosen a poor combination of options or done something else in a manner not conducive to optimal performance. Nonetheless, PETSc does not appear to be the best choice for this application.

5.3 Parallelization Using PESSL

Because of the poor results obtained from the PETSc library, a second attempt was made to parallelize the matrix solver portion of the code using IBM's Parallel Engineering and Scientific Subroutine Library (PESSL). PESSL is a library of numerical subroutines that are optimized for the IBM RS/6000 architecture, leading us to expect superior performance on the Cornell Theory Center SP2. Furthermore, it is compatible with the widely available ScaLAPACK library [17], thus facilitating ports to additional architectures.

The use of PESSL is relatively straightforward. It is able to operate on the data structures created in the original MPI version of our code without modification. It is necessary, however, to set up descriptors giving the global dimensions, block structure, and other partition information for each data object. We found that the method used to partition the matrix had a great impact on performance and, therefore, experimented with several methods. The task of finding an optimal partitioning scheme and block size was hindered, however, by the sheer number of modifications to the code required to implement different partitioning schemes such as slab, cyclic, and block cyclic.

5.3.1 Slab Partitioning

Initially, the data was partitioned exactly as in the MPI version of the code; that is, with each processor holding one contiguous block of rows of the moment matrix. Such a partitioning scheme was very easy to implement, since the MPI matrix fill routines could be used without modification. It was only necessary to set up the PESSL data descriptors for the matrix and vectors and then subsequently to call the LU factorization and solve routines PZGETRF and PZGETRS.

Unfortunately, this method exhibits the drawback of being slow, although it is far better than PETSc. Figure 5 shows the speedup achieved in terms of wall clock time performance of the solve routines for a system of 1000 unknowns. It shows that the PESSL implementation scales better than both the simple implementation and the PETSc implementation. However, the slab partitioning approach imposes a load imbalance on the LU factorization process. To improve the performance of the parallel LU factorization, an alternative partitioning scheme needs to be employed.

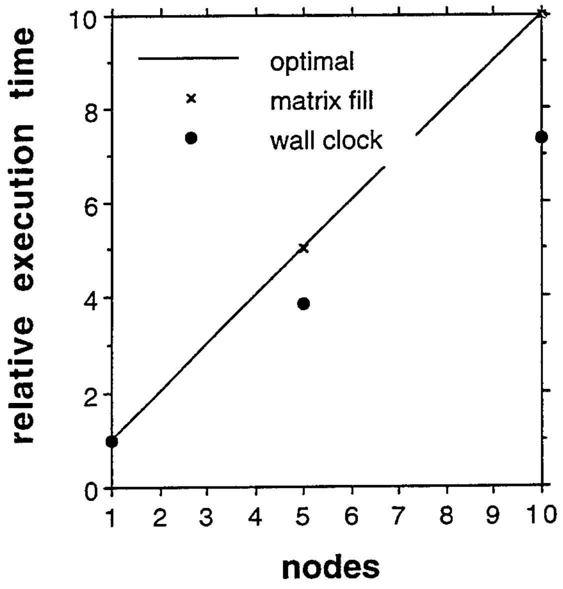


Figure 5. Speedup obtained for a system of order 1000 when the PESSL library was employed with slab partitioning.

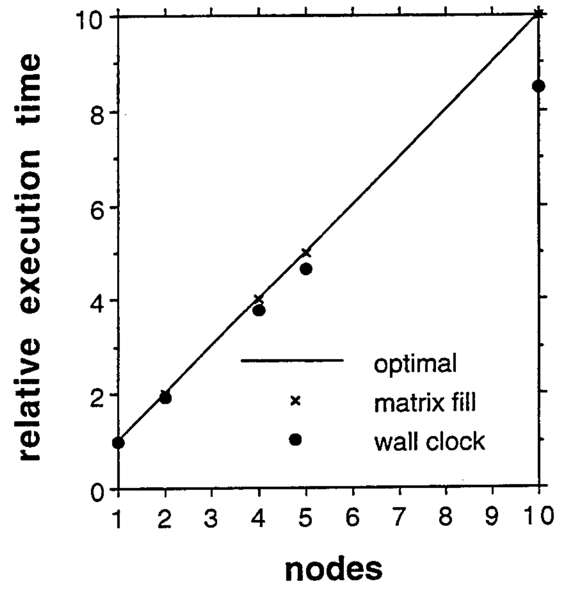


Figure 6. Speedup obtained for a system of order 1000 when the PESSL library was employed with cyclic partitioning.

5.3.2 Cyclic Partitioning

In an attempt to compensate for the deficiencies of the slab partitioning method, a second approach was taken. This time, rows were distributed across processors cyclically with a block size of two rows. For example, if there are two processors, blocks 1, 3, 5, 7, etc. reside on the first processor, while blocks 2, 4, 6, 8, etc. reside on the second processor. Such a scheme helped to balance the processor load and, as a consequence, produced much better results (Figure 6).

The performance obtained using cyclic partitioning could likely be improved by using a larger block size, which would reduce the communication overhead and probably provide greater parallel speedup. The block size of two rows was chosen for convenience, since in the version of the code in which only the matrix fill is done in parallel, row elements are computed in groups of two.

5.3.3 Results Obtained Using Cyclic Partitioning

In an attempt to determine the ability of the modified application to scale to large problems, we executed it on systems consisting of 3060, 5520, and 10240 unknowns. The cyclic PESSL version of the program was altered so that the coefficient matrix is dynamically allocated to only use the space needed on a given node, making the solution of these systems possible. A 40-node SP machine was used to solve a system with 10240 unknowns, requiring about 1.9 hours wall clock time to execute. To demonstrate scalability, we ran the program for a 5520-unknown system using both 10 SP nodes and 20 SP nodes. The wall clock time of the execution on 20 SP nodes is almost one half of the wall clock time of the 10 SP node case.

6. Conclusion

This paper described the parallelization of the 3D IE–AP code developed by Aberegg [6]. The IE-AP procedure tends to shift the computational burden away from the matrix solve part of the process and toward the matrix fill part, suggesting that it might be well-suited for parallel-architecture implementation. The IBM RS6000 SP2 was used as a parallel-architecture platform for the implementation.

The matrix fill task was easily parallelized using available MPI instructions. Two implementations of the matrix solve process were explored. While the PETSc implementation did not result in overall speedup, the PESSL implementation did exhibit an improvement over a serial implementation. Timing results are presented to demonstrate the performance.

The parallel-architecture implementation of the IE–AP approach will facilitate systematic testing and evaluation of the formulation on a wide range of scattering targets. Future work will address the overall efficiency of the IE–AP approach.

7. References

[1] G. C. Fox, M. A. Johnson, G. A. Lyzenga, S. W. Otto, J. K. Salmon, and D. W. Walker, *Solving Problems on Concurrent Processors*, Englewood Cliffs, New Jersey: Prentice-Hall, 1988.

[2] T. Cwik and J. Patterson, *Computational Electromagnetics and Supercomputer Architecture*, PIER 7, Cambridge, Massachusetts: EMW Publishing, 1993.

[3] X. Shen, *Massive Parallel Processing Applied to Computational Electromagnetics*, PhD Dissertation, Syracuse University, 1994.

[4] X. Shen, G. E. Mortensen, C. C. Cha, G. Cheng, and G. C. Fox, "Parallelization of the Parametric Patch Moment Method Code," *Digest of the 1995 ACES Symposium*, Monterey, CA, March 20-24, 1995.

[5] K. R. Aberegg and A. F. Peterson, "Application of the integral equation–asymptotic phase method to two-dimensional scattering," *IEEE Trans. Antennas Propagat.*, vol. 43, pp. 534-537, May 1995.

[6] K. R. Aberegg, *Electromagnetic scattering using the integral equation–asymptotic phase method.* PhD Dissertation, Georgia Institute of Technology, Atlanta, 1995.

[7] K. R. Aberegg, A. Taguchi, and A. F. Peterson, "Application of higher-order vector basis functions to surface integral equation formulations," *Radio Science*, vol. 31, pp. 1207-1213, September-October 1996.

[8] A. F. Peterson and K. R. Aberegg, "Parametric mapping of vector basis functions for surface integral equation formulations," *ACES J.*, vol. 10, pp. 107-115, November 1995.

[9] A. F. Peterson, "The interior resonance problem associated with surface integral equations of electromagnetics: Numerical consequences and a survey of remedies," *Electromagnetics*, vol. 10, pp. 293-312, 1990.

[10] A. F. Peterson, S. L. Ray, and R. Mittra, *Computational Methods for Electromagnetics*, New York: IEEE Press, 1998.

[11] D. I. Kaklamani and A. Marsh, "Benchmarking high-performance computing platforms in analyzing electrically large planar conducting structures via a parallel computed method of moments technique," *Radio Science*, vol. 31, pp. 1281-1290, Sep.-Oct. 1996.

[12] T. Cwik, "Parallel Decomposition Methods for the Solution of Electromagnetic Scattering Problems", *Electromagnetics*, Vol. 12, pp. 343-357, 1992.

[13] J. P. Brooks, K. K. Ghosh, E. Harrigan, D. S. Katz, and A. Taflove, "Progress in CRAY-based algorithms for computational electromagnetics," in T. Cwik and J. Patterson, *Computational Electromagnetics and Supercomputer Architecture*, PIER 7, Cambridge, Massachusetts: EMW Publishing, 1993.

[14] S. D. Gedney, A. F. Peterson, and R. Mittra, "The moment method solution of electromagnetic scattering problems on MIMD and SIMD hypercube supercomputers," in T. Cwik and J. Patterson, *Computational Electromagnetics and Supercomputer Architecture*, PIER 7, Cambridge, Massachusetts: EMW Publishing, 1993.

[15] D. S. Scott, "Solving large out-of-core systems of linear equations using the Intel iPSC/860," in T. Cwik and J. Patterson, *Computational Electromagnetics and Supercomputer Architecture*, PIER 7, Cambridge, Massachusetts: EMW Publishing, 1993.

[16] T. Cwik, R. van de Geijn, and J. Patterson, "Application of massively parallel computation to integral equation models of electromagnetic scattering", J. Opt. Soc. Am. A, Vol. 11, No. 4, pp. 1538-1545, April 1994.

[17] ScaLAPACK User's Guide may be found online at: http://www.netlib.org/scalapack/slug/scalapack_slug.html

ACES JOURNAL, VOL. 13, NO. 2, JULY 1998, SI: CEM & HPC

Performance Optimization of an Integral Equation Code for Jet Engine Scattering on CRAY-C90

Mikhail Smelyanskiy
Department of Electrical
Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48109-2122

Edward S. Davidson
Department of Electrical
Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48109-2122

John L. Volakis
Radiation Laboratory
Department of Electrical
Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48109-2122

Abstract

The numerical solution of Maxwell's equations is a computationally intensive task and use of high-performance parallel computing facilities is necessary for the larger class of practical problems in scattering, propagation and antenna modeling. It is therefore necessary to carefully consider algorithm optimizations aimed at improving the code's run time performance on the computing platform employed. Although some performance improvement can be derived from compiler-level optimizations, further speed-up may involve manual effort in algorithm restructuring, data layout , and parallelization. This paper focuses on the manual optimizations used to improve the performance of a moment method code for the analysis of a cylindrically periodic structure, as is the case with a jet engine. We describe the steps taken which resulted in nearly two orders of magnitude improvement over the original version of the code. A 16-processor shared-memory CRAY-C90 vector supercomputer was employed. Our optimization took advantage of SSD its solid-state storage, enabled better loop vectorizations, parallelized the *matrix_fill* routine, and called appropriate CRAY-C90 library routines.

1 Introduction

The application of analytical and numerical techniques in conjunction with advanced architecture and software has allowed more accurate simulations of very large electrical systems. In this paper we consider the parallelization and optimization of a code for evaluating the Radar Cross-Section (RCS). This is a method of moments (MoM) code specifically applied to jet engine scattering and exploit the inherent cylindrical periodicity of the engine geometry to reduce computation. As is the case with all MoM codes, a dense system of equations results from the discretization of the jet engine blades using the usual subsectional surface basis functions [1]. This code solves the dense matrix system via LU decomposition.

Typically, the most dramatic speed-up after code optimization is achieved by concentrating on the matrix fill and solution step (LU decomposition or the inverse FFT employed in the iterative solver) of the code. We therefore concentrate on optimization techniques which emphasize performance improvements for these steps and are aimed at reducing both CPU and wall-clock time.

This paper is organized as follows. Section 2 reviews the general concepts of performance optimization and simple examples are included to demonstrate the crucial role of optimization. Concepts of parallelization are discussed in section 3. Vectorization and parallelization techniques are applied to the MoM codes of interest in section 4 and the resulting speed ups are shown, following a short introduction to vectorization. Further optimizations of the MoM code are mentioned in the conclusion.

2 Optimization Overview

The first step in optimizing a computer algorithm is to evaluate its overall performance. This stage is aimed at identifying the critical sections of the code that consume most of the execution time and various profiling tools may be used to gather related runtime statistics. Figure 1 shows an example of the output from the CRAY-C90 flowtrace profiling tool applied to the unoptimized original version of the MoM code. The routines which require most of the most execution time are:

DYADFREE: calculates the free space dyadic Green's function.
FSGCYL: part of the general double surface integration routine called from the Inverse FFT routine.

VECT_DIFF: called mostly from the matrix fill routine to find the centroidal distance between source and observation triangles, to compute the edge lengths of the triangles, etc.
I_PARAMS: used to calculate the geometrical parameters for the closed-form integration routine which computes the potential integrals for uniform and linearly varying surface sources distributed on a planar polygon. This integration routine is part of the matrix fill routine.
AC_INTCYL: part of the general, double surface integration routine called from the matrix fill routine.

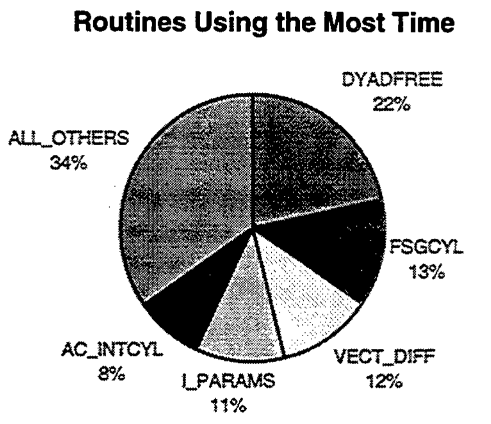


FIGURE 1: Runtime Profile of the most dominant routines

Having determined the most dominant sections, requiring the largest fractions of time and/or resources, effective code optimization can begin. Applying an optimization technique may involve the following steps (See Figure 2).

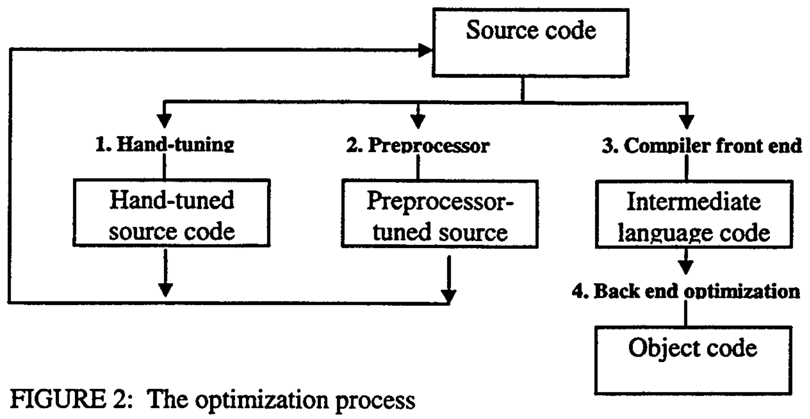


FIGURE 2: The optimization process

At the hand-tuning stage the user performs a number of optimizations at the source-code level. Examples of such optimizations include reordering programming statements or expressions and changing the memory access patterns of loops [2, 3]. Next, an optimizing preprocessor, if available, takes the source code and performs transformations enabled by user-selectable switches. Typical examples are dead code elimination, inlining, interprocedural analysis, library-call generation, etc. The output source code is also optimized to take advantage of architectural features of the host system. Then the output source code is submitted to the compiler whose front end translates it into intermediate language (IL). Its back end optimizer then translates the IL code into machine language and in this process it may apply a wide range of optimizations at the IL level, depending on the user-selectable flag settings which invoke specific sets of optimizations.

The tuning strategy should, of course, be aimed at producing the maximum performance gain with minimum effort. For the example, if we increase the performance of DYADFREE by 20% (see Fig. 1), the program will improve by 4.5% because DYADFREE takes 21.8% of the total time. If we also speed-up FSGCYL by 20%, the overall improvement will be 2.5%. Thus, sections of code that take the highest percentage of CPU time should be investigated first. It is of course important that the numerical results be checked after each optimization step to ensure the correctness and preciseness of the code after the reordering operations.

The sufficiency of the overall program performance is subsequently assessed either intuitively, or more formally, e.g. by using different performance characterization tools which provide bounds on the achievable performance of the code [4, 5]. A performance bound hierarchy model may successively include the effects of machine peak performance, high level application workloads, compiler-inserted overhead, compiler generated instruction schedule, cache effects, etc., and may be applied to loops, procedures, sections or entire codes. If later effects can be reduced or eliminated, performance may be made to approach earlier bounds which represent the potential performance of the application, if the effects of all later levels are eliminated.

Once a program has been sufficiently optimized for a single processor, the next step is to assess whether the application code can take advantage of multi-processor computing platforms. If so, the application is then parallelized and optimized for parallel execution.

2.1 Optimization Examples

It is hard to overstress the importance of optimization. The examples below demonstrate some simple techniques that can significantly improve code performance and speed-up. Details on the employed techniques themselves and further examples may be found in [2, 3].

Example1: (An Array-Processing Example)
Consider the two codes shown in Figure 3 which perform element-wise array multiplication [6]. These codes are clearly seen to be functionally equivalent.

```
do i=1,n
   do j=1,n
      c(i,j)=c(i,j)+a(i,j)*b(i,j)
   enddo
enddo
            (a) stride_n.f
```

```
do j=1,n
   do i=1,n
      c(i,j)=c(i,j)+a(i,j)*b(i,j)
   enddo
enddo
            (b) stride1.f
```

FIGURE 3: The Array Multiplication. (a) Original. (b) Loop interchanged

The only difference between the two examples in Figure 3 is that the array elements are referenced in a different order. All runs were made on an IBM RISC System/6000, Model 530 with a 64KB cache. The arrays were all declared REAL*8. A timing loop was inserted around the loops in the examples so that the reported time is the average of 50 million inner loop iterations. Figure 4 shows this time (in microseconds), as a function of n.

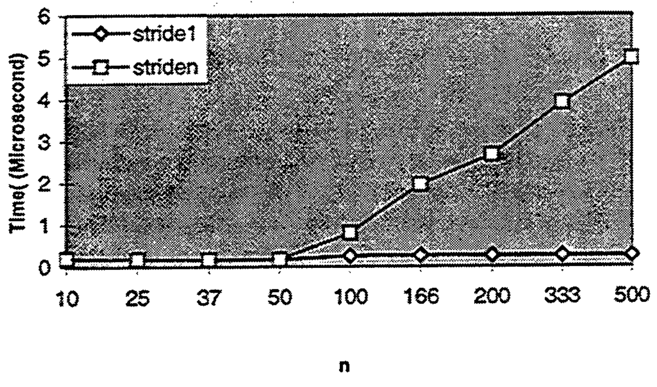


FIGURE 4: Performance on a RISC Sytem6000 Model 530 with 64 KB data cache

As seen in Figure 4, the performance differs significantly between the two codes. For small n, there is little difference in performance, but as *n* grows, **stride1** runs significantly faster then **stride_n**. In FORTRAN, arrays are stored in "column major order", implying that the leftmost subscript changes more rapidly as memory-adjacent elements are accessed. In the **stride1** routine, successive iterations of the inner loop access array elements that are adjacent in memory. That is the array elements are accessed in the same order as stored in memory.

However, in **stride_n** successive iterations of the inner loop access array elements that are stored in memory *n* entries apart (one array column) in memory. In this case the arrays are said to be accessed with stride *n*. When a single element is read into the processor, adjacent elements (comprising one "cache line") are automatically brought into the high-speed cache memory along with it. The user has no choice regarding this automatic procedure of cache loading. Clearly, if all entries brought into the cache are soon referenced(as in **stride1**), there is a memory access delay only for the first element in each cache line that the processor reads in. However, if other entries in this line are referenced much later (as in **stride_n**), the line with the referenced entries may get replaced in the cache before they are referenced; referencing an element that is in the cache is called a *cache hit*, otherwise the reference is a *cache miss* and suffers a delay called the *miss penalty*. The advantage of the **stride1** code is that there is roughly one miss per cache line of elements accessed, whereas almost every access to an element is a miss in **stride_n** code – unless *n* is small enough so that entire array fits in cache and remains there indefinitely. This scenario is easily seen in Figure 4, where both **stride1** and **stride_n** versions take the same time to run for $n \le 50$: 3 arrays of $50 \times 50 \times 8$ bytes = 60 KB < Size of the Cache (64KB). Obviously, an understanding of the machine's cache structure is important in writing code routines that have the best potential for optimum performance.[1]

Example 2: (Matrix Multiplication Example)

As another example, let us consider the three codes below (see Figure 5) which contain different matrix multiplication algorithms. All three codes produce the same end mathematical results. The differences are in the loop index order and the procedure used for the execution of the matrix multiplications. The first two algorithms differ in the loop index order and the third one takes advantage of matrix block multiplication.

```
do i=1,n
   do k=1,n
      do j=1,n
         c(i,j)=c(i,j)+a(i,k)*b(k,j)
      enddo
   enddo
enddo
```

(a) IKJ formulation

```
do j=1,n
   do k=1,n
      do i=1,n
         c(i,j)=c(i,j)+a(i,k)*b(k,j)
      enddo
   enddo
enddo
```

(b) JKI formulation

```
do ii=1,n,nb                 ! blocking loop
   do jj=1,n,nb              ! blocking loop
     do kk=1,n,nb            ! blocking loop
       do i=ii,ii+nb-1       ! loop within block
         do j=jj,jj+nb-1     ! loop within block
          s=c(j,i)
          do k=kk,kk+nb-1    ! loop within block
            s=s+a(j,k)*b(k,i)
          enddo
          c(j,i)=s
       enddo
     enddo
   enddo
enddo
```

(c) Block formulation

FIGURE 5: Three Formulations of Matrix Multiplication

[1] Although CRAY has no cache, its memory is divided into multiple banks with cycle time >> 1. Problems similar to those described in *Example 1* occur if not all banks are referenced, i.e. if GCD(stride, number of banks) $\neq$ 1. GCD (Greatest Common Divisor) is always 1 if stride = 1.

The performance of the three codes in Figure 5 is illustrated in Figure 6 based on runs made on a RISC System 6000 workstation with a 64 KB cache. As can be seen the *IKJ* algorithm exhibits the worst performance because successive iterations of the inner loop access elements of arrays *b* and *c* with stride *n*, resulting in multiple cache misses. Accesses to *b* and *c* are stride 1 in the JKI formulation, resulting in much fewer misses; however for large *n*, the arrays do not fit in cache and some misses still result from the fact that array elements have been replaced in the cache before they are rereferenced. Blocking is a technique for large arrays to reduce cache misses in nested array-processing loops. This is done by processing the data in blocks or strips which are small enough to fit in the cache. The principle behind blocking is that for every array element brought into the cache we wish to perform as many of the computations as possible on that element before it is forced out of the cache by other program actions. The blocking formulation of matrix multiplication algorithm with a blocking factor of 50 has much better performance (for models with a 64KB cache) than the JKI or IKJ formulations for large arrays. We note that the CPU timing for the fourth routine, **esslp2,** shown in Figure 6 refers to measurements of the engineering/scientific library subroutine (**essl**) for performing matrix multiplication. This library has been hand-coded to take maximum advantage of instruction overlap in the machine [6]; details of such proprietary libraries are, however, unavailable to users.

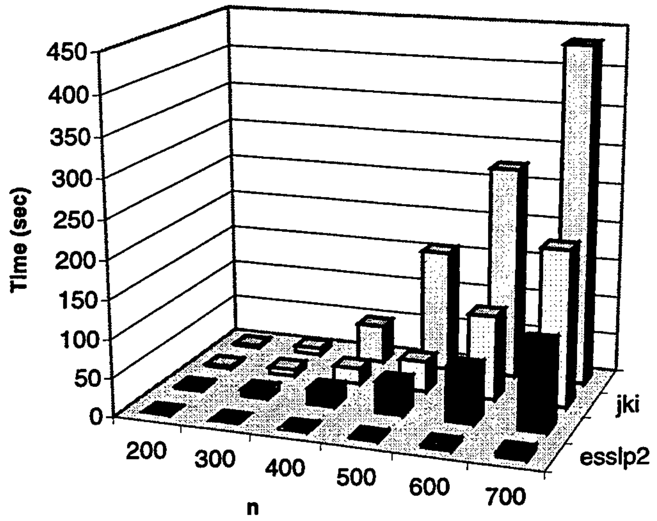
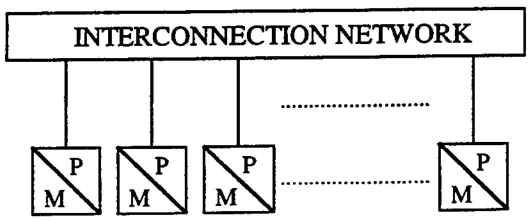
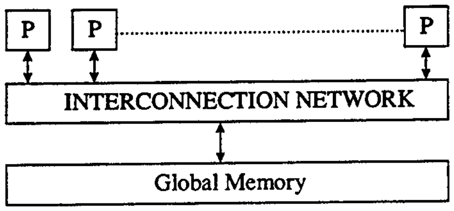


FIGURE 6: Total Run Time of Different Matrix Multiplication Formulations.

3 Basic Concepts of Parallelization

Once a program is sufficiently optimized on single-processors (relative to some goal), the next step is to find if parallelization can be used to speed up the operations/algorithms that consume the most CPU time. Many algorithms are serial on some levels and parallel on others. For instance, an iterative system solver can be parallelized at the matrix-vector product level to scale its performance, even though these solvers are inherently serial at the highest (outermost) iterative level. When parallelizing a code, one must consider the type of multi-processor platform available. ***Message passing*** and ***shared-memory*** multi processor architectures require different parallelization paradigms. In the message-passing (distributed-memory) architecture, see Figure 7(a), each processor has its own local memory space and communication among the processors is done by sending explicit messages through the interconnection network. The SP2 is typical of distributed-memory architecture. In the typical shared-memory





(a) (b)

FIGURE 7: Two Parallel Processors Interaction Paradigms.. (a) Message-Passing (b) Shared Memory

architecture, see Figure 7(b), all processors share the same global memory space. In this case, the processors communicate implicitly simply by accessing the same memory, which allows them to share code and data.

Example 1 (Matrix-Vector Product Example for Shared-Memory machine):
As mentioned above, the most critical (time consuming) operation in an iterative solver is the matrix-vector product. The serial version of a code for carrying out the MATRIX-VECTOR product is given in Figure 8(a). This sequential algorithm requires n^2 multiplications and additions resulting in $O(n^2)$ runtime complexity.

```
procedure MAT_VECT(A, x, y)
begin
  do i:=1 to n
     do j:=1 to n
  S:     y(i):=y(i)+A(i,j)*x(j);
     end do
  end do
end MAT_VECT

(a)
```

```
procedure MAT_VECT(A, x, y)
begin
  do all i:=1 to n
    do j:=1 to n
      y(i):=y(i)+A(i,j)*x(j);
    end do

  end do
end MAT_VECT

(b)
```

FIGURE 8: Matrix-Vector Product. (a) Serial Algorithm. (b) Shared-Memory Implementation.

One way of expressing parallelism on a shared-memory system is through the parallelization of the iteration loop. Basically, each iteration can be divided among the processors and executed concurrently (in parallel). Full parallelism is achieved if a sufficient number of processors is available. If there are fewer than *n* processors, some or all processors may execute two or more iterations. For the serial (i.e. uniprocessor) program in Figure 8 (a), consider a particular value *i* and the execution of the statement *S* for the iterations *j=1, 2...n* of the innermost loop:

y(i) = y(i) + A(i, 1)*x(1)
y(i) = y(i) + A(i, 2)*x(1)
...

As seen, the value assigned to *y(i)* in the first iteration is used in the second and, thus, there exists a "loop-carrier dependence" from *S* to *S* preventing parallel execution of multiple *j* loop iterations. This dependence could be broken by computing *p* subtotals for *y(i)* on *p* processors and then summing these subtotals in parallel. However, this modification of the code would violate Fortran precedence rules and thus may alter the numerical precision of the results. In contrast, the *i*-loop can be executed in parallel by rewriting the code as in Figure 8(b). In this case there are no conflicts between outermost iterations of the *i*-loop; that is, element *y(i)* is assigned and used on that iteration only. A sequential do-loop that has no conflicts is called doall-loop. Suppose *n* processors are available for the execution of the doall-loop. The compiler will then assign the work in such a way that each processor executes the highlighted portion of the code for one value of *i* in Figure 8 (b). Whereas each element of y is computed sequentially, different elements are computed in parallel. For p processors where $p < n$, the compiler will distribute the *y* elements to the processors as equitably as possible. Hence, the peak performance of the doall-loop on *p* processors $(p \leq n)$ results in $O(n^2 / p)$ run time complexity for the code in Figure 8(b).

4 Moment Method Code Optimization on CRAY-C90

4.1 Moment Method Fundamentals

The method of moments (MoM) is one of the standard approaches for the solution of a surface and volume integral equations. This solution technique can be applied to a metallic, homogeneous dielectric as well as inhomogeneous [7]. The boundary integral equation is of the form

$$\oiint_{\partial S} \bar{J}(\bar{r}')G(\bar{r},\bar{r}')\partial s' = \bar{f}^{i}(\bar{r}) \tag{1}$$

where $\bar{J}(\bar{r})$denotes the unknown current density, $G(\bar{r},\bar{r}')$ is the pertinent Green's function and $\bar{f}^{inc}(\bar{r})$ defines the excitation. Note that the parameters $\bar{r}'$and $\bar{r}$ refer to position vectors for the integration and observation points, respectively. The latter is the location at which the integral equation is enforced. In the context of the moment method, the observation point is placed at a set of discrete points over the illuminated structure to yield a corresponding number of equations. Introducing the subsectional expansion

$$\bar{J}(\bar{r}) = \sum_{n=1}^{N} I_n \bar{f}_n(\bar{r}) \tag{2}$$

for the surface currents, where $\bar{f}_n(\bar{r})$ is the expansion basis over the *nth* pair of triangles [1], we obtain a discrete system of equations for the solution of the unknown coefficients I_n. We typically write this system as

$$[Z]_{mn}\{I_n\} = \{V_n\} \tag{3}$$

where $[Z]_{mn}$ is the $N \times N$ impedance matrix whose elements are computed from

$$Z_{mn} = \oiint_{S_m} \bar{f}_m(\bar{r}) \cdot \oiint_{S_n} \bar{f}_n(\bar{r}')\, G(\bar{r},\bar{r}')\partial s \partial s' \tag{4}$$

and

$$V_m = \oiint_{S_m} \bar{f}_m(\bar{r}) \cdot \bar{E}^{inc}(\bar{r})\partial s \tag{5}$$

where *m* and *n* are indices on the edges of the surface facets S_m (S_n) denoting the surface area over the *mth* testing (*nth* integration) triangle pair. Because of the Green's function [8],

$$G(\bar{r},\bar{r}') = Ae^{-jk|\bar{r}-\bar{r}'|} / |\bar{r}-\bar{r}'| \tag{6}$$

where *A* is a constant and k is the wavenumber, the matrix *[Z]* is fully populated and this is a distinct feature of the moment method. The most popular and straightforward approach is to perform an LU decomposition of *[Z]* and then solve for *{I_n}*. As noted above, this is one of the most time consuming steps of the code and must be optimized for optimum CPU performance. For our specific application of jet engine scattering, a last step is the computation of the fields at a plane in front of the engine and the generation of the modal scattering matrix [9]. The modal scattering matrix is then used for the propagation of the fields through the inlet duct and the computation of the scattered fields due to a given excitation field.

In the following subsection we describe the optimization of the moment method code for jet engine scattering on a 16 processor, shared-memory CRAY-C90 vector supercomputer. An interesting aspect of this code is the use of a cylindrically periodic Green's function which takes advantage of the jet engine blade periodicity. As a result, the computational domain is reduced to a domain over a single blade and this yields a reduction in the number of unknowns by a factor equal to N_s, the number of blades (typically 20 to 30). However, to take advantage of blade periodicity, it is necessary to decompose the excitation field into cylindrical duct modes and to loop through all of the modes which are allowed to propagate unattenuated [10, 11, 12]. The scattered modes and subsequently the corresponding fields due to each mode is then computed by carrying out an inverse cylindrical Fourier transform. Although, this modal decomposition may be more expensive for single angle computations, this disadvantage disappears when multiple excitations are considered. Most importantly, the mode by mode excitation is particularly attractive for parallelization since the computation for each mode excitation is independent of the others.

4.2 MoM Code Structure

A flowchart of the jet engine moment method code is given in Figure 9. This flow chart shows

- the outer loop going through all cylindrical modes
- the computation of the matrix element Z_{mn}
- system solution via LU decomposition
- inverse cylindrical FFT operation for generating each row of the scattering matrix

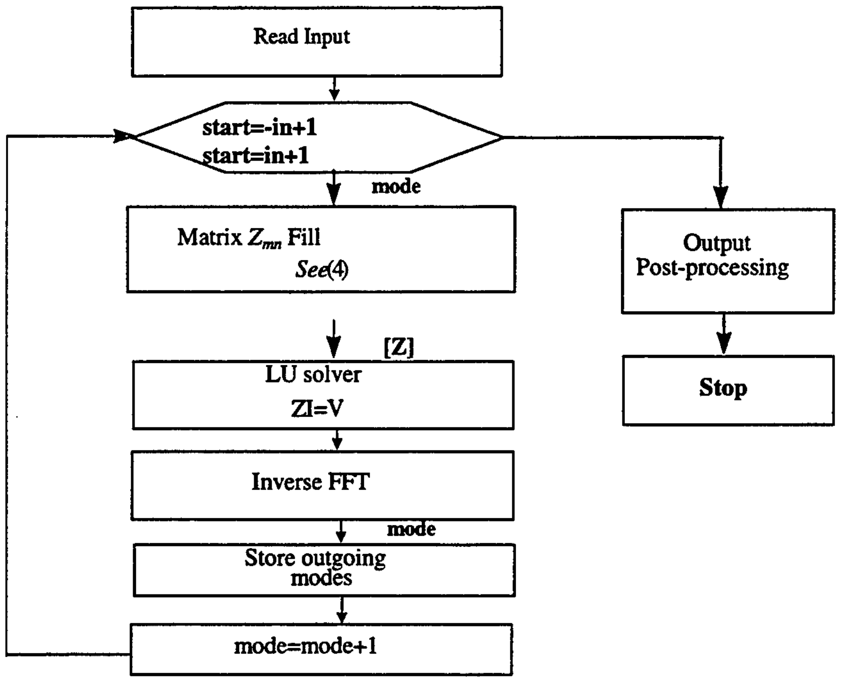


FIGURE 9: MoM Code Main Iteration Loop.

The matrix fill and Inverse FFT routine pseudo-codes are shown in Figures 10 (a) and (b) respectively. In these codes, N_s denotes the number of engine slices/blades and *const* is

```
for all slices ( s=0, N_s-1)
    for all source triangles ( p=1, num_source_triangles)
        for all observation triangles (q=1, num_observation_triangles)
            compute next entry of
            impedance matrix $[A]^{(s)}$
        endfor
    endfor
    $[Z]^{(ms)} = [Z]^{(ms)} + const(m, s)*[A]^{(s)}$
endfor
```
(a)

```
for all (angles i1=1, exrho)
    for all (point i2=1, ngas)
        for all (angles i3=1, exphi)
            for all (points i4=1, ngas)
                call integration routine
            endfor
        endfor
    endfor
endfor
```
(b)

FIGURE 10: (a) Matrix Fill Pseudo-code. (b) Inverse FFT pseudo-code.

a complex function of the mode number (m) and slice number(s). Also, the matrices $[A]^{(0)}$, $[A]^{(1)}$, ..., $[A]^{N_s-1}$ are the submatrices comprising the matrix *[Z]* and provide the interaction between the engine blades. Based on the previous discussions, the parallelization of this algorithm can be performed at several levels. For example, since the mode iterations of the main loop are entirely independent, they can be done in parallel. However, in this case each processor will be required to keep a full copy of the *[Z]* matrix and thus the storage requirements become very large. Also, load imbalance will become a significant bottleneck if number of modes is not evenly divisible by the total number of processors. To avoid these issues, we instead proceeded to individually parallelize the matrix fill, LU factorization and InverseFFT routines, thus taking advantage of the finer-grain paralellism of the code at this level.

4.3 Major Optimization Stages

Our optimization of MoM code consisted of four steps:

1. Porting of the matrix fill routine to static storage device (SSD[2])
2. Rewriting and parallelization of the matrix fill routine
3. Library call substitutions
4. Vectorization and parallelization of the inverse FFT routine

Sections 4.3.1 through 4.3.5 describe the optimization procedures with Section 4.3.2 giving a concise introduction vectorization.

4.3.1 Porting Matrix Fill Routine to Static Storage Device (SSD)

As shown in Figure 9, the main iteration loop calls the *matrix_fill* routine for every mode. In the original version of the code, each impedance submatrix $[Z]^{(s)}$ had to be recomputed for every slice (Figure 10(a)). This was done to avoid in-core storage of each $[Z]^{(s)}$ submatrix, even though the submatrices were identical from one slice to the other (since the slice is a periodic cell). The CRAY-C90 however provides way to avoid both excessive storage requirements and the need for a recomputation of the submatrices is to by making use of the SSD at the first call of the *matrix_fill* routine. At each subsequent mode loop, the stored matrices are recalled from the SSD to accumulate the sum in $[Z]^{(ms)}$ matrix. The resulting pseudo-code is shown in Figure 11. This change alone resulted in an 11x program speedup as seen below in Table 1.

```
for all (slices $s=0, N_s-1$)
  if (m==1) then                                        ! if first mode
    for all (source triangles p=1, num_source_triangles)
      for all observation triangles q=1, num_observation_triangles
        compute next entry of
        impedance matrix $[Z]^{(s)}$
      endfor
    endfor
    store $[Z]^{(s)}$ on SSD
  else
    read $[Z]^{(s)}$ from SSD                           ! avoid computing $[Z]^{(s)}$
  endif
  $[Z]^{(ms)} = [Z]^{(ms)} + const(m, s)*[Z]^{(s)}$
endfor
```

FIGURE 11: Pseudo-code to compute the impedance matrix using SSD storage.

4.3.2 Principles of Vectorization

Since a high level of vectorization is a must for efficient utilization of the CRAY-C90 processor, based on vectorization, in this section we give a brief introduction to the principles of vectorization before introducing the code speed-ups.

[2] SSD-solid state storage is very high speed secondary memory of up to 2048 Mwords (16Gbytes). CRAY-C90 supports up to 4 1800-Mbytes/s channels for accessing.

A vector is a set of scalar data items (elements), all of the same type. A vector instruction applies an arithmetic or logical operation to vectors by using *vector registers* as operand registers. A vector load/store instruction moves a vector between a vector register and memory with some constant memory strides. However if GCD(stride, number of memory banks) $\neq 1$, then the load/store may take twice as long, four times or more. There is also some support for irregular, nonconstant strides. However, a data layout in memory that allows stride=1 will always be fastest.

Vectorization converts scalar operations such as multiplications and additions to corresponding vector operations. That is, a vectorized code processes the multiplications/additions as a sequence of vector statements and this is particularly attractive for multiplying arrays in do-loops. A well-vectorized code can easily achieve a speed-up of 10 to 20 times, as compared to the equivalent scalar code. Vectorizing compilers perform the mechanics of vectorization; but they are much more effective if the code is written in a style that helps the compiler recognize vectors. They also report what was successfully vectorized and provide some hints about problems that prevented vectorizations. These hints are often useful for rewriting the code for more effective vectorization. Below we consider a few examples which illustrate the implementation of vectorization steps.

Examples of vectorization:

Example 3: (Recurrence)

Consider now the loop shown in Figure 12(a). The statement instances are executed in the order shown in the unrolled code. Note that the value of each *A(i)*, $2 \leq i \leq n$, is dependent on the *A(i-1)* calculated by the immediately previous statement (iteration). If we were to vectorize L, then the products *A(I-1)*B(i)* in the vector statement *A(1:n)=A(0:n-1)*B(1:n)* would be computed using the values *A(0:n-1)* that existed when entering the loop. Since incorrect results would be produced, a vectorizing compiler would indicate that this loop can not be vectorized due to its (loop-carried) dependence.

```
serial loop                      vectorization not possible
L:   do i=1, n                   L(1): A(1) = A(0)*B(1)
       S: A(i) = A(i-1)*B(i)     L(2): A(2) = A(1)*B(2)
     end do                        ...
              (a)                L(1): A(n) = A(n-1)*B(n)
                                          (b)
```

FIGURE 12: Vectorization example 4.

Example 4: (Loop Interchange)

However, consider the operation shown in Figure 13, referred to as the two-point difference scheme. One way to rewrite the loop as a set of vector statements is to interchange the loop order and this is done in Figure 13(b). Now, the inner loop may be converted to a vector statement, as shown in Figure 13(c).

```
serial loop                                  transformed loop
L:  do i=1, n                                L:   do j=1, n
      do j=1, n                                     do i=1, n
        S: A(i, j)=(A(i, j-1) + A(i, j+1))/2          S: A(i, j)=(A(i, j-1) + A(i, j+1))/2
      end do                                        end do
    end do                                        end do
                  (a)                                           (b)
Vectorized loop
L:  do j=1,n
      A(1:n, j)=(A(1-n, j-1)+A(1:n, j+1))/2
    end do
```

FIGURE 13: Vectorization example 4. (a) original code (b) loop restructuring before vectorization (c) vectorized code.

Example 5: (Loop Distribution)

In the loop shown in Figure 14(a) the operation S1 depends on the result of S2 in the previous iteration. This apparent interdependence, can be removed by distributing the loop as shown in Figure 14(b). The rewritten code, although not as compact, is now amenable to vectorization as shown in Figure 14 (c).

```
serial loop
L:   do i=1, n
       S1: D(i)=A(i-1)*D(i)
       S2: A(i)=B(i)+C(i)
     end do
            (a)
```

```
transformed loop
L21:  do i = 1, n
        S1: A(i)=B(i)+C(i)
      end do
L22:  do i = 1, n
        S1: D(i)=A(i-1)*D(i)
      end do
            (b)
```

```
vectorized loop
A(1:n)=B(1:n) + C(1:n)
D(1:n)=A(0:n-1)*D(1:n)
            (c)
```

FIGURE 14: Vectorization example 5. (a) original code (b) rewritten code (c) vectorized code

For code such as examples 4 and 5 if a vectorizing compiler is unable to vectorize the loop, performing the transformation in (b) manually should allow the compiler to produce the vector code shown in (c).

4.3.3 Vectorization and Parallelization of the Inverse Transform

Runs made with the original version of the MoM code revealed very short average vector lengths. Also, very little improvement in this regard was observed when the compiler optimization flags were enabled. Such a poor utilization of vector registers is explained by the fact that the original version of the MoM code was written for RISC processors. Therefore no attempt was made to program the code in a manner suitable for vectorization (or parallelization). After careful examination of the code, both manually and using various performance tools, we discovered that the Inverse FFT (or IFFT) integration routine had the highest potential for vectorization. The original code fragment took a single source triangle and passed it sequentially through several different subroutines to perform a single surface integration over each vector function as shown in Figure 15.

```
subroutine get_vertices(vert,q)
   real vert(3,3)
   integer q
   vert(1,1)=...
   ....
   vert(3,3)=...
end get_vertices

subroutine integrate(vert, vint)
   real vert(3,3), vint(3)
   vint(1)=...
   vint(2)=...
   vint(3)=...
end integrate

real vert(3,3)     ! list of vertices of the triangle
real vint(3,3)     ! result of  integration
for each source triangle q
  call get_vertices(vert,q)
  call integrate(vert, vint)
  ...
endfor
```

FIGURE 15: Inverse FFT Integration Routine.

This version of Inverse FFT also yielded a poor performance on the CRAY-C90. More specifically, the average vector length was 6, with 128 being the theoretical maximum limited by the length of the processor's vector registers. Further, automatic vectorization failed to perform the interprocedural analysis required by this code.

Our first successful phase of optimization was to replace these single element calls with a call to subroutines that accepted as inputs the entire list of integration elements rather than providing them one by one. This transformation resulted in a rather efficient vectorizable code (See Figure 16). This code permits vectorization within each low-level subroutine, thereby eliminating the need for interprocedural analysis.

```
real vert(num_triangles, 3, 3)          ! list of vertices of the triangle
real vint(num_triangles, 3, 3)          ! result of integration

call get_vertices(vert)
call integrate(vert, vint)

subroutine get_vertices(vert)                    subroutine integrate(vert, vint)
  real vert(num_triangles, 3, 3)                   real vert(num_triangles, 3, 3)
  for each triangle q                              real vint(num_triangles, 3, 3)
    vert(q, 1, 1)                                  for each triangle q
    ...                                              vint(q, 1)=...
    vert(q, 3, 3)=...                                vint(q, 2)=...
  endfor                                             vint(q, 3)=...
end get_vertices                                   endfor
                                                 end integrate
```

FIGURE 16: Vectorizable Inverse FFT Integration Routine.

Further optimization included standard techniques such as subroutine inlining (both manually and automatically with the inline compiler option), loop splitting, scalar expansion, etc., all leading to an average vector length increase by an additional factor of 10 or so.

The next step was to parallelize the IFFT routine. Since CRAY-C90 is a parallel shared-memory machine, the loop distribution principles discussed in *Example 2* apply here. We observe that the individual calls to the integration routine (see Figure 10(b)) are entirely independent and, thus, the outermost loop can be readily distributed yielding the parallel code in Figure 17. Finally, we note that maximum parallelization is achieved by maintaining good load balance, and this is readily done provided that the outer loop trip count of divisible by the total number of processors.

```
do all angles i1=1, exrho
   for all point i2=1, ngas
      for all angles i3=1, exphi
         for all points i4=1, ngas
            call integration routine
         endfor
      endfor
   endfor
end do all
```

FIGURE 17: Parallel Inverse FFT Integration Routine

4.3.4 Parallelization of MatrixFill Routine

Having achieved sufficient performance on one processor, the next optimization step is to parallelize the *matrix_fill* routine. As noted above the code uses a faceted surface model employing triangular elements for the characterization of the jet engine blade geometry (or slice) resulting in the discrete system given by (1).

Lets us begin with the serial version of *matrix_fill* as given in Figure 18(a). The code loops over the surface patches and computes the partial integrals for each of the integrals associated with currents flowing through each of the three edges that make up the patch. These three evaluations provide contributions to the matrix entries and are assembled later at their appropriate locations in the matrix *[Z]*. To parallelize the code we need to distribute the outermost loop source triangles to different processors as shown in Figure 18(b)[3]. Since, the three edges for both the source and observation triangles will not generally be assigned to the same processor, the GUARDING code block is needed to guarantee that only one processor updates a *[Z]* matrix entry at any one time.

[3] Distribution of loop iterations to multiple processors is called Multitasking in general. On CRAY-C90 this transformation can, in many cases, be automatically performed by the compiler, and is thus referred to as *Autotasking*.

```
for all observation triangles p
  for all source triangles q
    ...
    call integration routine
    for all edges m(1), m(2), m(3) of p, v
      for all edges n(1), n(2), n(3) of q, w
      ...
      $Z(m(v), n(w)) = Z(m(v), n(w)) + const(p, q, v, w)$
      ...
      endfor
    endfor
  endfor
endfor

                    (a)
```

```
do across all observation triangles p
  for all source triangles q
    ...
    call integration routine
    for all edges m(1), m(2), m(3) of p, v
      for all edges n(1), n(2), n(3) of q, w
      ...
GUARD mutex region
      $Z(m(v), n(w)) = Z(m(v), n(w)) + const(p, q, v, w)$
END GUARD
      ...
      endfor
    endfo
  endfor
end do across            (b)
```

FIGURE 18: Matrix Fill. (a) Serial version. (b)Parallel version.

The synchronization overhead to execute the guarding code is negligible compared with the time needed to go through other statements in the loop. Since, in general, the number of source triangles is much larger than the number of processors available, the above parallel version of *matrix_fill* should be quite efficient.

4.3.5 Library Call Substitutions

Where possible, the CRAY-C90 library routines were substituted into the original code. Most of these routines are carefully hand-tuned for both vector and parallel execution and it is therefore a good practice, in general to make use of the system library routines whenever possible. Of primary interest to us is the LU factorization routine which ranks second in the list of the most time-consuming operations in the MoM code. The original code employed the LINPACK routine ZGEFA and this was replaced by the equivalent CRAY-C90 CGETRF vector parallel routine. The speedup resulting from substitution of the CGTERF is shown in Figure 19, where all times are wall-clock times. It is clearly seen that substantial speedup was achieved. More specifically, the CRAY-C90 library LU routine reduced the CPU time by 30% percent on a single processor, and from 35 sec down to 5 sec on the 16-processor.

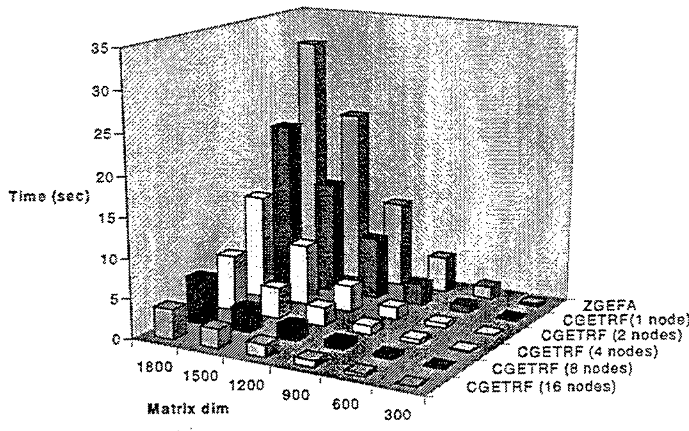


FIGURE 19: Scaling of CRAY-C90 matrix Factorization Routine

4.4 Performance of Individual Optimization Stages

The overall improvements after each optimization stage of the code, as described above, are summarized in the Table 1. The underlying geometry was a 21 engine blade geometry, 3 wavelengths in diameter.

		COMPILER OPTIMIZATION ONLY	SSD OPTIMIZATION	INVERSE TRANSFORM VECTORIZATION	FILL AUTOTASKING AND LIBRARY CALLS
FILL	CPU TIME	222	12.5	12.5	12.5
	WALL CLOCK	322	18	18	4
INVERSE TRANSFORM	CPU TIME	6.5	6.5	4	4
	WALL CLOCK	9	9	2.25	2.25
LU	CPU TIME	1.7	1.7	1.7	1.4
	WALL CLOCK	2.5	2.5	2.5	0.4
TOTAL	CPU TIME	235	21	19	18.3
	WALL CLOCK	340	30	23	6.68
	SPEED-UP		11.3	1.3	3.4
			50 TIMES TOTAL SPEEDUP		

TABLE 1: MoM Speedup after each of the Four Code Optimization Stages

The first three columns of the Table 1 present the run-time improvements of the serial code, whereas the fourth column indicate speed-up of the paralellized code which was run on the number of available processors. The first column shows the code run time in the absence of any hand-tuning with only compiler limited vectorization, inlining, etc. This time is comparable to the run time of the original code on a high performance workstation with compiler optimizations.

4 Conclusion/Further Optimizations

In this paper we described various optimization techniques that can be routinely used to improve the performance of standard moment method codes for electromagnetic applications as well as more specialized codes. We detailed several vectorization techniques which proved effective in tuning the performance of the MoM code on a CRAY-C90. We showed parallel loop distribution for *matrix_fill* and parallelization of the LU factorization. Although, *matrix_fill* scales as n^2 whereas LU factorization scales as n^3, the *matrix_fill* routine remains the most dominant part of the code for large values of n due to the large constant in front of n^2. In fact the crossover between $O(n^2)$ and $O(n^3)$ has not been reached [13].

Although, we successfully employed SSD storage to avoid redundant *matrix_fill* for every mode iteration, this approach is not likely to succeed outside the CRAY-C90 environment. We are currently working on porting the *matrix_fill* section of the code to a distributed memory architecture. In this case, each processor will be responsible for computing and storing only its assigned portion of the matrix entries. We hope that by carefully decomposing the mesh and mapping the triangular elements to different processors, we will be able to reduce the overall communication needs and thus achieve a scalable distributed memory code.

Acknowledgment

The results and code performance given in this paper were carried out at the Wright Laboratories Aeronautical Systems Major Shared Resource Center (ASC MSRC) computing facilities.

References

[1] S.M. Rao, D.R. Wilton, and A.W. Glisson. "Electromagnetic scattering by surfaces of arbitrary shape" IEEE Trans. Antennas Propagat., vol. AP-30, no3, pp. 409-418, May 1982.

[2] H.P Zima and B. Chapman. *Supercompilers for Parallel and Vector Computers. Frontier Series*, ACM Press, New York, NY, 1991

[3] Michael Wolfe. *High Performance Compilers for Parallel Computing*. Addison-Wesley, Reading, MA, 1997.

[4] Eric L. Boyd, Waqar Azeem, Hsien-Hsin Lee, Tien-Pao Shih, Shih-Hao Hung, and Edward S. Davidson. "A Hierarchical approach to modeling and improving the performance of scientific applications on the KSR1," Proceedings of the International Conference on Parallel Processing, August 94.

[5] William Mangione-Smith, Tien-Pao Shih, Santosh G. Abraham, and Edward S. Davidson. "Approaching a Machine-Application Bound in Delivered Performance on Scientific Code," Special Issue of IEEE Proceedings on Computer Performance Analysis, ugust 93.

[6] IBM Corporation, *Optimization and Tuning Guide for the Fortran and XL C Compilers.*

[7] R. F. Harrington. *Field Computation by Moment Methods*, Macmilian, New, York, 1968. (Reprinted by Krieger Publishing Company, Malabar, Florida, 1983).

[8] Johnson J.H Wang. *Generalized Moment Methods in Electromagnetics.* John Wiley & Sons, Inc., New York, 1991.

[9] H. T. Anastassiu, J. L. Volakis, and D. C. Ross. "The mode matching technique for electromagnetic scattering by cylindrical weveguides with canonical terminations," J. of Electromagnetic Waves and Applications, 9(11/12):1363-1391, Nov./Dec. 1995.

[10] D. C. Ross, J. L. Volakis, and H. T. Anastassiu, "Hybrid finite element-modal analysis of jet engine inlet scattering," *IEEE* Trans. Antennas and Propagation, 43(3):277-285, March 1995.

[11] D.c. Ross, J. L. Volakis, and H. T. Anastassiu, "Overlapping modal and geometric symmetries for computing jet engine inlet scattering," IEEE Trans. Antennas Propagation, 43(10):1159-1163, Oct. 1995.

[12] H. T. Anastassiu, J. L. Volakis, and D. C. Ross, and D. Andersh, "Electromagnetic scattering from simple jet engine models," IEEE Trans. Antennas and Propagation, 44(3):420-421, March 1996.

[13] Tom Cwik, Daniel S. Katz, "Scalable Solutions to Integral-Equatio and Finite-Element Simulations," IEEE Transactions on Antennas Propagation, vol. 45, no.3, March 1997

Optimisation and large scale computation in integral equation scattering analyses

S J Dodson, S P Walker[†], M J Bluck
Mechanical Engineering Department
Imperial College of Science Technology and Medicine
London SW7 2BX

Abstract : **The kinds of difficulties posed by large scattering computations change as larger problems are addressed. Unfavourable cost scalings make the performance of small, core, portions of code dominant, and require that they, and the overall code structure, be optimised for large scale computation. This is discussed in the context of rcs and scattering computations of multi-wavelength bodies using a time domain integral equation treatment. Examples presented include the NASA almond evaluated at 25 wavelengths long, and an assembly of 101 spherical scatterers of ~1/2 wavelength diameter each, occupying a volume of side ~250 wavelengths.**

1. Introduction

Many problems in CEM share the unattractive characteristic of having computational costs which increase sharply with problem (electrical) size[1], and transient scattering and radar cross section computations, on which this paper will concentrate, are a prime example of this. Whilst 'small' problems are thus not computationally difficult, something of a ceiling in electrical size is quickly encountered when tackling only slightly larger ones. Progress requires a judicious combination of frugal algorithm, recasting for large, and in practice massively parallel, computers, and careful coding and optimisation. Once very large problems are tackled, additional difficulties arise in what are otherwise peripheral issues; the areas of mesh generation, and results manipulation and display quickly become major computational tasks in their own right.

In this paper we will describe attempts to reduce the computational cost of integral equation time domain (IETD) analyses[2], using the above approaches, but with particular emphasis on practical techniques we have found effective in optimisation. Additionally, the issues and problems of mesh generation and results display will be considered.

In the remainder of this introduction we will summarise very briefly the algorithms being employed. Section 2 will discuss the various aspects of optimisation which arise in trying to use these on large problems, section 3 considers pre- and post-processing issues, and in section 4 we will give examples of the performance of the codes.

For scattering from a perfectly conducting body, subject to some incident wave, the surface field is given by[3]

$$2\pi\mathbf{H}(\mathbf{r},t) = 4\pi\mathbf{H}_{inc}(\mathbf{r},t) + \int_{\partial\Omega}\left(\mathbf{n}'\times\mathbf{H}(\mathbf{r}',t^*)\right)\times \frac{\hat{\mathbf{R}}}{R^2} + \left(\mathbf{n}'\times\frac{\partial\mathbf{H}}{\partial t}(\mathbf{r}',t^*)\right)\times\frac{\hat{\mathbf{R}}}{cR}ds' \qquad (1)$$

where **r** and **r'** are surface locations, $\mathbf{R} = \mathbf{r}' - \mathbf{r}$, wave speed is c, and time t and retarded time $t^* = t - R/c$. Most of what follows in this paper is independent of the details of the discretisation of (1) employed, but here we will use an implicit

[†*] Correspondence: s.p.walker@ic.ac.uk, +44 (0)171 823 8845 fax, +44 (0) 171 594 7058, http://www.ic.ac.uk/mechanics

curvilinear isoparametric approach. Details are rather tedious[4], but eventually a matrix equation for the field at the next timestep is obtained, with the field expressed as a weighted sum of surface fields over the rest of the body at times up to one transit time (W timesteps) ago. For discretisation with N nodes i and j, at timestep k+1, this new field is given by

$$2\pi \mathbf{H}_i^{k+1} - \sum_{j=1}^{N} \alpha_{i,j}^{0} \mathbf{H}_j^{k+1} = \tag{2}$$

$$4\pi \mathbf{H}_{inc,i}^{k+1} + \sum_{j=1}^{N} \sum_{w=1}^{W} \alpha_{i,j}^{w} \mathbf{H}_j^{k+1-w}$$

The coefficients α (3 by 3 matrices, later reduced to 2 by 2 by application of pec boundary conditions) characterise the influence of components of the historical field at j on components of the new field being sought at i. Note that an explicit treatment is obtained if the timestep is so small that the new fields at nodes adjacent to i do not influence the field at i. Otherwise, (2) represents a sparse matrix equation:

$$\mathbf{A} \cdot \mathbf{h}^{k+1} = \mathbf{c}^{k+1} \tag{3}$$

where $\mathbf{h}^{k+1}$ is $(\mathbf{H}_1^{k+1},..., \mathbf{H}_N^{k+1})^T$, and the vector **c** results from evaluation of the summations on the right of (2).

The main components of the computational work, our primary interest here, are clear from (2) and (3). For a given discretisation, the number of nodes will vary with f^2, so the work of forming the coefficients α will scale with f^4. Cost of formation of each **c** thus scales with f^4, and with typically the number of timesteps required being roughly proportional to electrical size, we obtain the usual f^5 cost scaling of IETD approaches. Storage costs of α scale with f^4, and of the field history (nodes x timesteps) with f^3. Fuller discussion of these scaling issues is given by Miller in the paper cited earlier[1].

Illumination with a pulse which is short compared to the body size results in fields which are small over most of the body most of the time. This can be exploited by (a) integrating in (1) over only those portions of the body where the field was significant at the relevant retarded time, and (b) only evaluating the field at locations where it is expected to be found to be significant. With the fraction of the body surface over which significant fields obtain declining roughly with frequency, these two approximations together reduce the scaling of cost with frequency by up to two powers. A fuller description of this, and an investigation of the reduced accuracy and associated cost reductions, has been presented earlier[5,6]. It is on the implementation and optimisation of this modified algorithm that this present paper will concentrate.

2. Optimisations

2.1 Memory and operation tradeoffs

For the conventional IETD the main storage requirement, the coefficients α, scales with f^4. These can be formed once at the beginning of the job and then multiplied by historical values to form the right hand vectors at every timestep. This we will term the 'in-core' method. The size of this matrix is normally ~6 x 2 x 2 x N x N words. For example, for a 1,202 node NASA almond[7] (approximately 4 wavelengths long) the matrix occupies 133Mb if stored (as it is) in single precision. This is sizeable but not impossible on a workstation. However, for say a ~25 wavelength case (such as we will analyse later) the storage required is about 327 Gb; ~ten times the entire core of the largest of supercomputers.

This storage strategy would limit problem sizes on say a 1 Gb workstation to ~3,200 nodes, or an almond ~6.5 wavelengths long. This job would however take only ~1/2 hour to form the matrix, and ~1 hour to perform the timestepping, making the storage clearly the limiting factor.

An alternative approach is to generate the matrix coefficients afresh at every timestep, use them and discard them. The overall scalings are the same, but the costs are increased by a constant factor of rather more than an order of magnitude (the cost of forming the coefficient, relative to 'using' it in the multiplications of (2)). The memory savings are considerable, with the now dominant history scaling with f^3. For the same 3,200 node almond this requires only ~34 Mb.

Repeated reading of the coefficients from disk, where the provision of tens of Gb is no real problem, is an alternative approach. For a workstation this could be practicable and quicker. However, large problems require large computers, which in practice means parallel computers, and the speedup in input of such machines is generally far below their increase in processing speed, making the recalculation far faster.

Similar arguments cause repeated recalculation of the matrix coefficients to be the best approach for the modified algorithm. This is reinforced by the fact that the approximations in the approach result in much of the matrix never actually being used and hence never calculated.

History storage for the modified algorithm is much reduced, as only the history of 'active' periods is stored. Being a handful of pulse durations worth at any one node, it has a cost scaling with f^2. This is the same scaling, and indeed is lower in amount than, the storage requirements of the mesh data itself. For example, storage requirements of the 25 wavelength almond analysed later using this approach were about 300 Mb. The history alone in the conventional approach would have required 200 Mb; the history in the modified approach was ~50Mb.

2.2 Parallelisation

The whole area of parallelisation, covering domain decomposition, minimising of communications, and load balancing, is large. There has been work on frequency domain integral equation treatments[8,9], but little has been published[10,11] on time domain integral methods. This latter was for the conventional IETD approach. There are significant changes required to exploit massively parallel machines properly using the modified algorithm. A paper detailing these has recently been written[12], and we will summarise the approach here.

Storage of the same data on all processors must be avoided, and this is done by partitioning the mesh by elements, with their associated mesh and history data, over processors. In the modified approach only a subset of 'field' nodes i equation (2) are involved at each timestep, from each of which integration is performed over a subset (different for each i, and differing at each timestep) of 'boundary' nodes j. Each processor is caused to integrate from every relevant i over such boundary nodes j as it is custodian of. However, the nature of the propagation of the pulse is such that both these groups of nodes tend to be spatially fairly contiguous. Given the careful node numbering optimisation schemes built into most CAD packages, they tend also to be fairly contiguous in terms of node number. Deliberate randomisation of node numbering is required to achieve good load balancing, but when this is done very nearly equal work is performed on each processor. With very little interprocessor communication required, overall parallelisation is very effective.

2.3 Profiling

Code development, debugging and so on naturally tends to be performed using small test problems. Whilst different portions of the code may be known in theory to have different cost scalings, fixed overhead costs normally dominate till sizeable problems are tackled. It is only on the largest of jobs that the portions of the code having asymptotically dominant costs become clear. Equally, of course, it is only if the intent is to use the code for such jobs that the effort of such identification and optimisation is warranted. A further discouragement is that the resulting optimisations, to be of most use, are often machine specific, driven say by the particular pipelining and cache arrangements of a particular processor.

The technique of 'profiling' is invaluable in identifying those portions of a code where most effort is being expended. It is often the case that a tiny fraction, by lines of code, is found to account for the vast majority of the execution time. Sometimes, an optimised library routine may be available, for a 'generic' activity such as matrix-vector multiplication, and this can be substituted. Alternatively, manual optimisation can bring significant gains.

We would comment in passing that in our experience such optimisation should be confined strictly to those portions of the code which are found to be truly costly. The kinds of optimisation we will describe shortly almost invariably make the code more complicated, and harder to read and understand. Introduction of temporary scalar variables, loop unrollings and so on, need copious in-code documentation to be comprehensible even to the developer himself only a short while later.

In the conventional IETD, run 'in core' it was suggested above that repeated formation of the right hand side of (2) was the dominant cost. This is confirmed by the profile of figure 1. The routine performing this (pecformrhs) accounts for over 90% of the cpu cycles. The second most expensive, amatmult, is part of the iterative solver used in repeated solution of the sparse matrix equation in (2). This profile was run for a 1,202 node, 4 wavelength body. For the small test cases on which the code was originally developed things are markedly different.

What matters changes greatly when we move to the modified algorithm, with repeated matrix coefficient calculation, but only for node-element pairs corresponding both to significant expected field values and significant historical (retarded) fields. Figure 2 shows the profile of this case, for the same 1,202 node, 4 wavelength test body. We see that the majority of the time is spent in a routine 'ipecnsq9_': the routine which integrates over a non-self element, on a pec body, discretised with a 9 noded quadrilateral element. What was dominant, the right hand side formation (routine p3formrhs), is now an order of magnitude smaller in relative importance.

Forms of profile like these are very attractive from the point of view of optimisation, with the main computational cost so localised, such that minor changes can provide significant gains. The routine ipecnsq9 for example, comprises only some ~200 lines of code, some 10 of which in turn account for the vast majority of ipecnsq9 time, in a package of ~14,000 lines.

2.4 Coding modifications

Modern optimising compilers themselves perform a variety of optimisations to make the code run faster. The level of optimisation which results in the best performance varies from program to program, and changing this compilation option can often result in run-time improvements. However, if the algorithm is poorly organised, any amount of optimisation is unlikely to result in efficient code.

The profile of the modified code shows that the majority of the time is spent in the non-self element integration routine. This routine integrates from a field node over on element by performing the following summations:

$$\alpha_i^m = \sum_{q_1}\sum_{q_2}\sum_{\beta}\sum_{\alpha} S_\alpha(\xi_{q_1},\eta_{q_2}) \tag{4}$$
$$\left[\frac{T_\beta(\tau)}{R^3} + \frac{\dot{T}_\beta(\tau)}{\delta t R^2}\right][A']\left|\mathbf{J}(\xi_{q_1},\eta_{q_2})\right|\omega_{q_1}\omega_{q_2}$$

These summations are over respectively the 9 spatial shape functions S , the three temporal shape functions T, and the two sets of Gaussian integration locations for the two dimensional surface integral.

One obvious and classical optimisation performed by compilers is to move expressions which are 'loop invariant' outside of the loop:

(a) Original Code

```
do I = 1, N
   do J= 1, M
      A(J,I) = A(J,I) + B(I) * C + D
* E
   end do
end do
```

(b) Optimised Version

```
temp = D * E
do I = 1, N
   temp1 = temp + B(I) * C
   do J = 1, M
      A(J,I) = A(J,I) + temp1
   end do
end do
```

For example, in (4) all the terms (except δt) in the above expression are dependent on Gauss point locations q_1 and q_2. Only S_α is dependent on α and only T_β and $\dot{T}_\beta$ on β. Therefore, we can rewrite (1) in a slightly different form as

$$\alpha_i^m = \sum_{q_1}\sum_{q_2}[A']\left|\mathrm{J}(\xi_{q_1},\eta_{q_2})\right|\omega_{q_1}\omega_{q_2}\frac{1}{R^2} \quad (5)$$

$$\sum_\beta\left[\frac{T_\beta(\tau)}{R}+\frac{\dot{T}_\beta(\tau)}{\delta t}\right]\sum_\alpha S_\alpha(\xi_{q_1},\eta_{q_2})$$

(with 1/R itself actually evaluated outside the summation, but this becomes cumbersome to show algebraically). For simple expressions most compilers will perform this optimisation automatically, and (we at least) commonly write code to reflect the most natural form of the algebra, assuming that the compiler will optimise later. However, for more complicated expressions such as (4) we have found it is much more effective to separate the terms by hand.

Another and often more important reason to breakdown each term into its dependencies is to determine the loop ordering. This is relevant in optimising memory references. For multi-dimensional arrays in FORTRAN, iterating the first subscript in a loop gives fastest memory access, with unit stride (sequential assessing of adjacent memory locations) optimising benefit gained from the cache. Therefore, we order the dimensions of our arrays according to the loop ordering in the routine which dominates the run-time. This means if an array requires (Gauss point, β,α) subscripts we then order the subscripts $A(\alpha,\beta,q_1,q_2)$ if optimisation requires the loops to be ordered as shown in (2). It is a farsighted developer who can foresee the details of loop ordering of dominant-cost routines at the early stage at which data structures are specified, and such optimisations can then require considerable retrospective book-keeping code modifications. This provides another strong incentive to confine them to truly costly code portions.

RISC workstations, such as the DEC alpha, can perform multiple instructions per clock cycle and can pipeline operations. This can allow further optimisations to be made. A classic example is the unrolling of loops to allow calculations from different iterations to be executed together.

These optimisations are well known and can be applied to a number of codes across platforms. Any tuning of a program will have to involve a combination of optimisations. For example, unrolling of loops, use of temporary scalars and so on. These slight changes to the code can result in the compiler interpreting the code differently. Some optimisations will be general, while others will work best on certain platforms. Clearly, there is a trade-off between run-time gained in optimising a code and time spent performing the optimisations. However, knowledge of the concepts of compiler optimisations can help in producing fast code and if the majority of the run-time is spent in a few lines of code, optimisation can be quick and effective.

2.5 Results of optimisations

In this section we present the results obtained by the optimisations discussed above. All optimisations were made on the non-self integration routine ipecnsq9. Relative timings for an entire run using the modified algorithm are shown in Table 1.

TABLE 1
Relative CPU time for two different NASA Almond meshes with different code optimisations.

Code Options (see key for code descriptions)	1,202 Nodes	5,282 Nodes
1. Original code	2.536	3.661
2. Unit stride in inner loop array	1.585	2.070
3. Optimum compilation level	1.526	2.034
4. Loop invariant calculations removed	1.136	1.130
5. Inner loop unrolled	1.000	1.000

1. Original ipecnsq9 compiled with the default optimisation level (-O4).
2. As 1, with the subscript ordering of the array which holds the integration weights modified for unit stride in the inner loop of the integration routine.
3. As 2, with a compilation level which should provide fast run-time (-O5 -fast)
4. As 3, with loop invariant calculations of β and α terms manually removed from the inner loop.
5. As 4, with inner loop (3 x 3 matrix scalar multiply) unrolled manually

This table shows clearly the benefits of progressive code optimisations; approaching a factor of four on the larger problem. In all cases only a few lines of the routine where altered. The most beneficial of all the optimisations was modification of the array in the inner loop to use unit stride lengths. The benefits of the optimisation increase with problem size, as an increasing fraction of total cost is incurred in the subject routine.

3. Pre- and post processing

Mesh generation is performed using commercially available CAD packages, primarily MacNeal Schwendler's Patran, and SDRC's Ideas. In both cases, the prime use of the CAD suite is for mechanical design, but the capabilities they offer are well suited to electromagnetic modelling. The families of elements we employ, such as 8 and 9 noded quadratic quadrilaterals, and 6 noded quadratic triangles, are supported as a matter of course. Facilities such as automatic mapped and paved mesh generation, mesh seeding, and local mesh refinement are of considerable use. Display of results is done in much the same way, with (say) the time dependent vector surface field fed back into the CAD package.

However, much as some optimisation issues only arise on large problems, various additional difficulties arise in the pre- and post-processing for large problems.

Some of the surface meshes we have generated approach 100,000 nodes in size. Since such CAD packages are primarily designed to generate three dimensional meshes for finite element stress analysis, such mesh sizes are well within their capability. They nonetheless require quite powerful workstations to manipulate and display them.

With quadratic modelling in both time and space, a rational (normalised) timestep would be equal to the largest nodal separation, and this is the criterion we would generally employ on modest sizes of problem. However, it became clear that on large bodies, it was common to generate (and indeed difficult not to generate) meshes with a very small number of elements rather larger than say 1/10 of a wavelength, or whatever nominal target the analyst had intended. The automated timestep selection was modified to examine the profile of the nodal separations, and to select a timestep corresponding to typically the 95th percentile nodal separation. Whilst a handful of percent of elements with perhaps double the intended nodal separations will not degrade the entire calculation to any significant extent, a timestep based on this would do so.

Sheer size of the results is also a problem. Using the conventional IETD algorithm for a 25 wavelength almond, for the ~3 transits necessary, generates an (ASCII) surface field history comprising some 1.3 Gb. Manipulation of files of this size is cumbersome, to say the least, and getting a CAD package to read, manipulate and display it is of itself a major computational task. One major practical by-product of the modified algorithm is that the results files, containing as they do the history only for the relatively brief active periods, are much smaller. The same 25 wavelength almond, for example, analysed thus generates a much more manageable result file of only 74 Mb.

Similar difficulties arise in calculation of the rcs. To do this efficiently requires that the surface result be stored in core. For the modified algorithm this is straightforward. For the conventional approach in the 25 wavelength case rcs calculation using the 1.3 Gb (ASCII) file would require about 1 Gb of memory.

4. Example results

4.1 NASA almonds

We will show some example results and timings for analysis of multi-wavelength NASA almonds.

Meshes comprising from 2,962 to 41,266 nodes were employed, each suitable for analysing (down to) a particular pulse width, and the extraction via fourier transform of results up to some particular frequency. Head or rear-on illumination was employed, but symmetry was not exploited.

In figures 3(a) and 3(b) are shown VV rcs calculations over a wide frequency range, with the almond varying from 1 to ~25 wavelengths long. The frequency range over which each mesh was employed is indicated on the figures. Also indicated on the figure are results computed elsewhere[7,13,14], and measured results[7], extracted by manual measurement from the published graphs. (This latter is practically a difficult process, and there is additional uncertainty introduced by this.) As is seen, the modified algorithm generally computes results in good agreement with the experiments and other computations, with reasonable consistency between meshes.

Figures 4 (a), (b) and (c) show harmonic surface fields, extracted via fourier transform, at different frequencies. As the frequency increases, the fields on the downstream, shadowed portion of the body are falling noticeably, as the regime where optical methods become applicable is approached. *[These are colour figures; please visit www.me.ic.ac.uk/mechanics/CWP/CWP_publications.html or www.emclab.umr.edu/aces/acesjrnl.htmlto view]*

With the CAD packages employed it is straightforward to construct and mesh a 'cutting plane', for the display of near field results. This is done in figure 5, for a fourier transform at a frequency corresponding to the almond being 25 wavelengths long. Near field evaluation was itself a sizeable computation, with 60,000 locations defined in the cutting plane, from each of which integration over the ~41,000 node body had to be performed, taking some 24 hours on a Dec alpha workstation. The near field results show clearly the rearwards shedding of the field, accounting for the low backscattered rcs.

[This is a colour figure; please visit www.me.ic.ac.uk/mechanics/CWP/CWP_publications.html or www.emclab.umr.edu/aces/acesjrnl.htmlto view]

As an example of timings, the 25 wavelength case took the equivalent of 2 1/4 hours on a 512 processor Cray T3D. Table 2 shows normalised timings for the various almond cases. Times are characterised by numbers of integrations (as seen earlier, by far the dominant cost), to eliminate inter-machine differences, and the table gives times relative the that for the 2962 case. The size in wavelengths associated with each mesh is the largest which can be extracted with consistent criteria regarding discretisation and timestep.

The 'scaling' shown is the power of frequency relating the time for the size in question to the next size smaller. Normal IETD cost scaling is with frequency to the fifth power. As is seen, in all cases the scaling is below 4, and it is falling as larger cases are considered. A knowledge of how many integrations were performed allows the cost relative to the conventional case to be accurately assessed; taking the 25 wavelength case as an example, the factor by which costs are lower is about 50.

TABLE 2. Relative timing for various NASA Almonds (meshes / lengths in wavelengths).

Nodes	w/l	Tip on		Tail on	
		Time	Scaling	Time	Scaling
2,962	6.9	1.0		1.0	
5,282	9.2	3.1	3.85	2.9	3.73
10,866	13.2	11.9	3.77	11.8	3.85
21,522	18.5	38.7	3.46	42.7	3.77
41,266	25.6	111.1	3.24	124.9	3.30

4.2 Multiple bodies

One advantage of integral methods is the absence of any need to discretise the free space between different scatterers. A simple example is a random assembly of 10 small spheres, each 2 units in diameter, occupying a circumscribing box of side ~100 units. These were illuminated with a pulse making the 'box' and individual spheres respectively about 25 and 1/2 wavelengths in size in terms of the extractable frequency.

Figure 6 shows conventional and modified IETD results for the surface field at an arbitrarily selected node on one sphere. Obviously there is no analytical result to compare this with, but the modified and conventional approaches are both seen to predict very similar fields. As the results indicate, any one sphere is essentially quiescent much of the time, awaiting the arrival of some reflected wave from elsewhere. The major saving of the modified approach in this case, which is by a factor of about 32 here, arises because no nugatory calculations are performed during this wait.

Figure 7 shows a larger version of a similar test problem, with now 100 spheres, and the box now about 250 wavelengths in side, with a total of ~10,000 nodes. The modified algorithm needed ~70 Mb of memory, and took ~48 hours on a workstation. This size of problem could not be run by us with the conventional IETD approach. The field history alone would have required about 1 Gb of memory, and the run would have taken some 720 times longer. The surface field at the three locations indicated in figure 7 is shown in figure 8.

5. Conclusions

Efficient algorithms, coupled with use of large parallel machines, and appropriate optimisation of code to suit large problems and large machines, have been shown together to make significant reductions in the computational cost of large scattering problems. This has been demonstrated by analysis of much larger bodies than has been reported previously using time domain integral methods.

References

1. Miller, E.K. A selective survey of computational electromagnetics. *IEEE Transactions on Antennas and Propagation* 36:pp1281-1305, (1988).

2. Gomez Martin, R., Salinas, A. and Rubio Bretones, A. Time-Domain Integral Equation

Methods For Transient Analysis. *IEEE Antennas and Propagation Magazine* 34(3):pp15-22, (1992).

3. Poggio, A.J. and Miller, E.K. Integral Equation Methods of Three-Dimensional Scattering Problems. In: *Computer Techniques for Electromagnetics*, edited by Mittra, R. Oxford: Pergamon Press, 1973, p. 159-265.

4. Bluck, M.J. and Walker, S.P. Time Domain BIE Analysis of Large Three Dimensional Electromagnetic Scattering Problems. *IEEE Transactions on Antennas and Propagation* 45:pp894-901, (1997).

5. Walker, S.P. Scattering analysis via time domain integral equations; methods to reduce the scaling of costs with frequency. *IEEE Antennas and Propagation Magazine* 39:pp13-20, (1997).

6. Dodson, S.J., Walker, S.P. and Bluck, M.J. Costs and cost scalings in time domain integral equation analysis of electromagnetic scattering. *IEEE Antennas and Propagation Magazine* (submitted):(1997).

7. Woo, A.C., Wang, H.T.G. and Schuh, M.J. Benchmark radar targets for the validation of computational electromagnetics programs. *IEEE Antennas and Propagation Magazine* 35:pp84-89, (1993).

8. Davidson, D.B. Parallel matrix solvers for moment method codes for MIMD computers. *Applied Computational Electromagnetics Society Journal* 8:pp144-175, (1993).

9. Cwik, T. Parallel decomposition methods for the solution of electromagnetic scattering problems. *Electromagnetics* 12:pp343-357, (1992).

10. Walker, S.P. and Leung, C.Y. Parallel computation of integral equation methods for three dimensional transient wave propagation. *Communications in Numerical Methods in Engineering* 11:pp515-524, (1995).

11. Walker, S.P. and Leung, C.Y. Parallel computation of time domain integral equation analyses of electromagnetic scattering and rcs. *IEEE Transactions on Antennas and Propagation* 45:pp614-619, (1997).

12. Dodson, S.J., Walker, S.P. and Bluck, M.J. Parallelisation issues for high speed time domain integral equation analysis. *Parallel Computing* submitted:(1997).

13. Volakis, J.L. Carlos-3D; A general purpose three dimensional method of moments scattering code. *IEEE Antennas and Propagation Magazine* 35:(1993).

14. Miller, E.M, Andersh, D.J and Terzouli, A.J.Jr Target facetisation level and the effect on Xpatch predictions. *9th Annual Rev Appl Comp Electromagnetics* pp610-617, (1993).

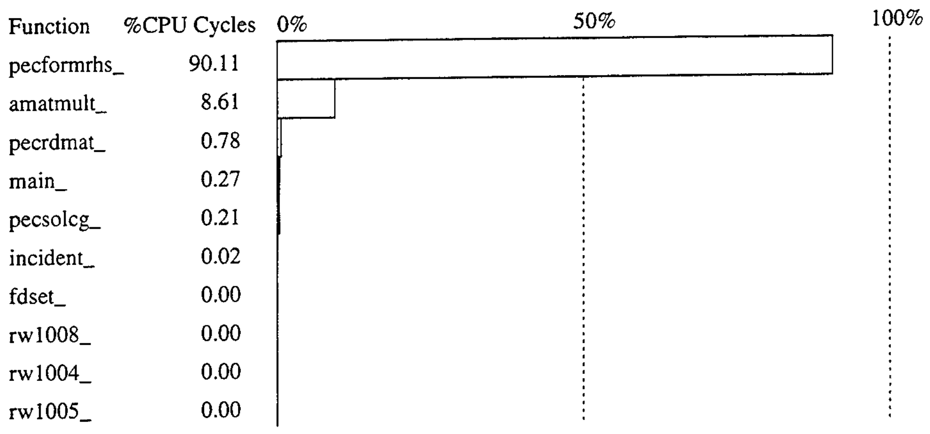


Figure 1

Profile for the conventional algorithm, 1202 node problem, matrix stored in-core. The bar chart shows the percentage of the CPU cycles spent in different subroutines. The dominant routine 'pecformrhs_' multiplies the matrix coefficients by historical field values to form the right hand side vector. The routine 'amatmult_' performs the iterative solution of the sparse left hand side [A] matrix.

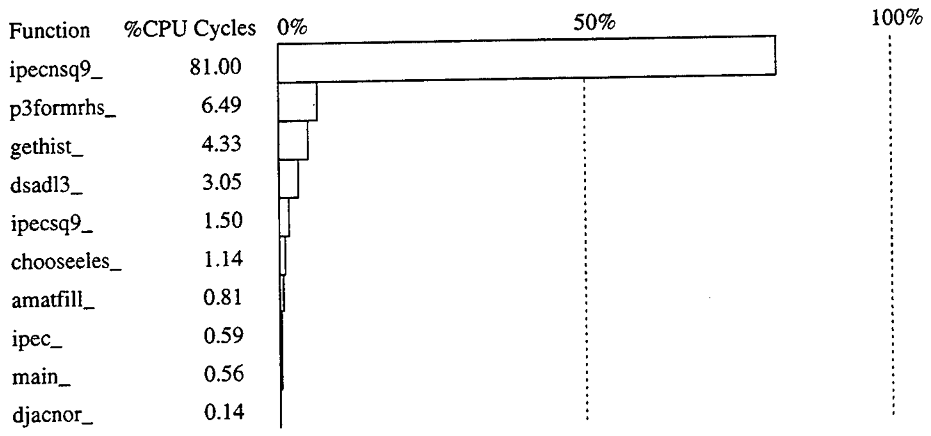
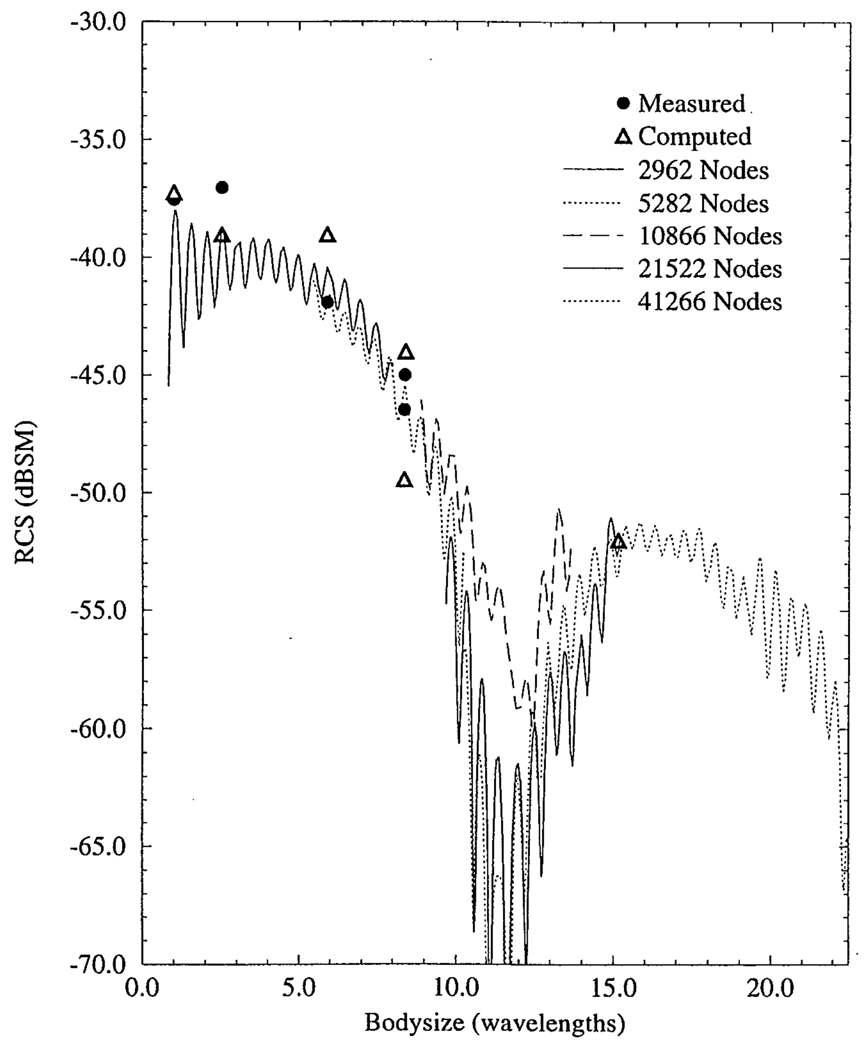


Figure 2

Profile for the modified approach, 1202 node problem, generating matrix coefficients as and when needed. The first five routines (ipecnsq9_ to ipecsq9_) form the matrix coefficients and multiply them by historical field values. The dominant routine, ipecnsq9_, forms the matrix coefficients for a non-self node / element pair. The optimisations shown in table 1 were performed on only this routine.



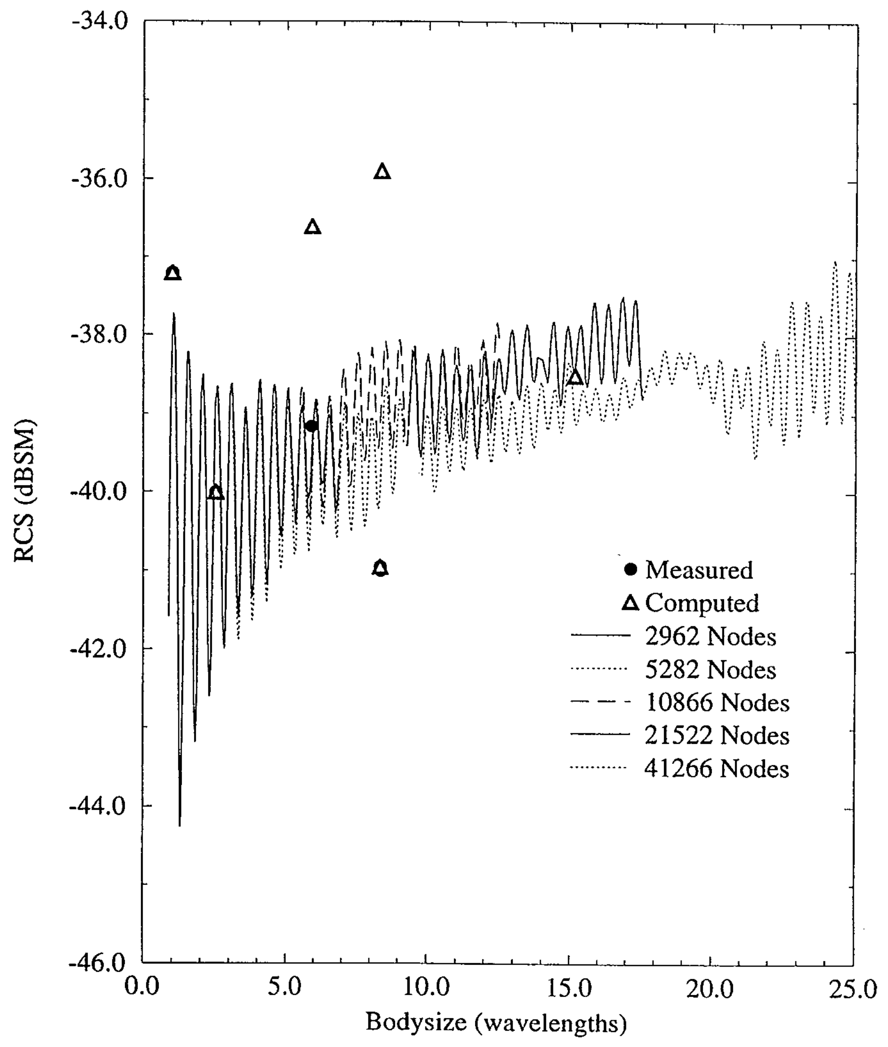
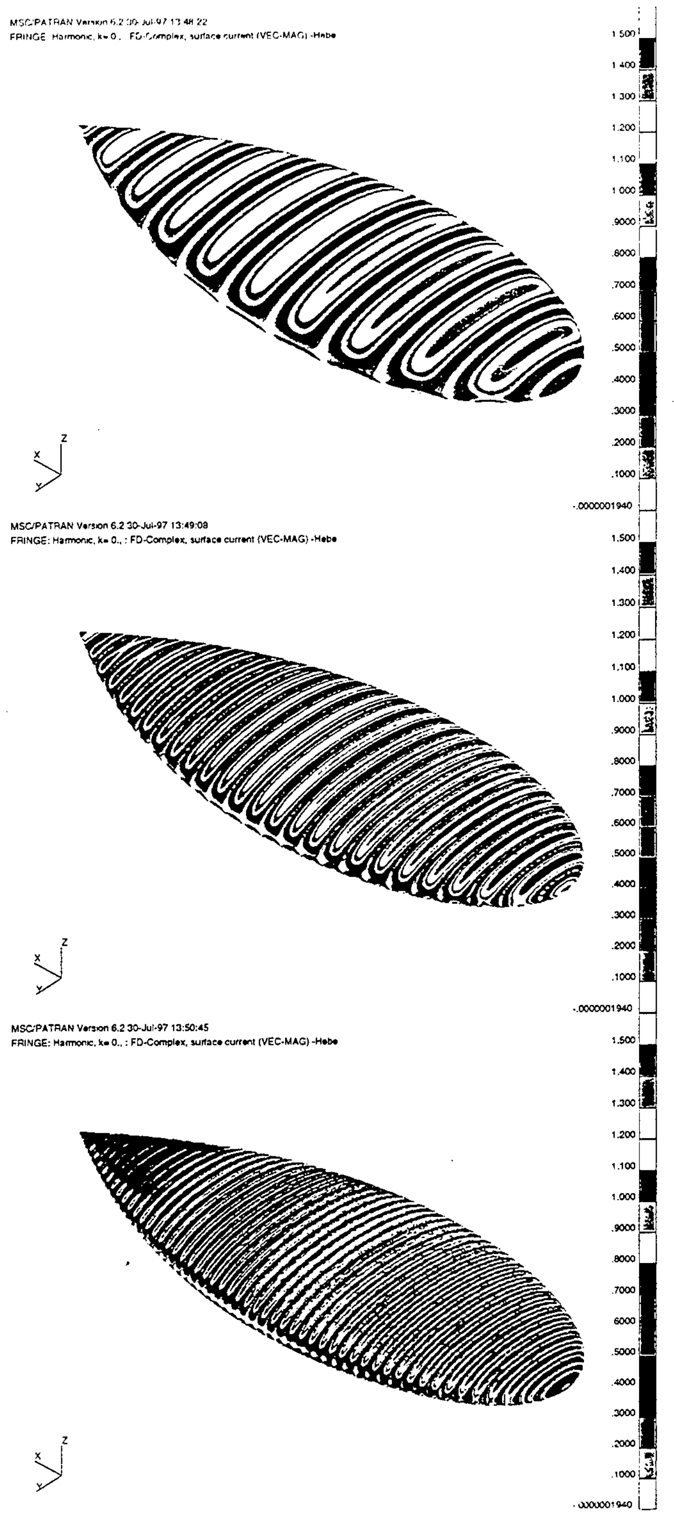


Figure 3

Backscatter versus bodysize for the 9.936" NASA almond. (a) represents propagation in the -x (tip-on) direction, and (b) propagation in the +x (end-on) direction. **E** polarisation is vertical (+z) in both cases. Frequency ranges for some meshes are shortened to improve the clarity of the graph.


MSC/PATRAN Version 6.2 30-Jul-97 13:48:22
FRINGE: Harmonic, k= 0., : FD-Complex, surface current (VEC-MAG) -Hebe
1.500
1.400
1.300
1.200
1.100
1.000
.9000
.8000
.7000
.6000
.5000
.4000
.3000
.2000
.1000
-.0000001940
MSC/PATRAN Version 6.2 30-Jul-97 13:49:08
FRINGE: Harmonic, k= 0., : FD-Complex, surface current (VEC-MAG) -Hebe
MSC/PATRAN Version 6.2 30-Jul-97 13:50:45
FRINGE: Harmonic, k= 0., : FD-Complex, surface current (VEC-MAG) -Hebe
x
y
z

Figures 4(a), (b) and (c)

Surface |**H**| extracted at (a) 5.67GHz, (b) 14.9GHz and (c) 30.4GHz, on the NASA almond. Propagation is $\pm x$ (broad end on), **E** vertical. Electrical lengths 4.77, 12.5 and 25.7 wavelengths respectively.

[These are colour figures; please visit www.me.ic.ac.uk/mechanics/CWP/CWP_publications.html or www.emclab.umr.edu/aces/acesjrnl.htmlto view]

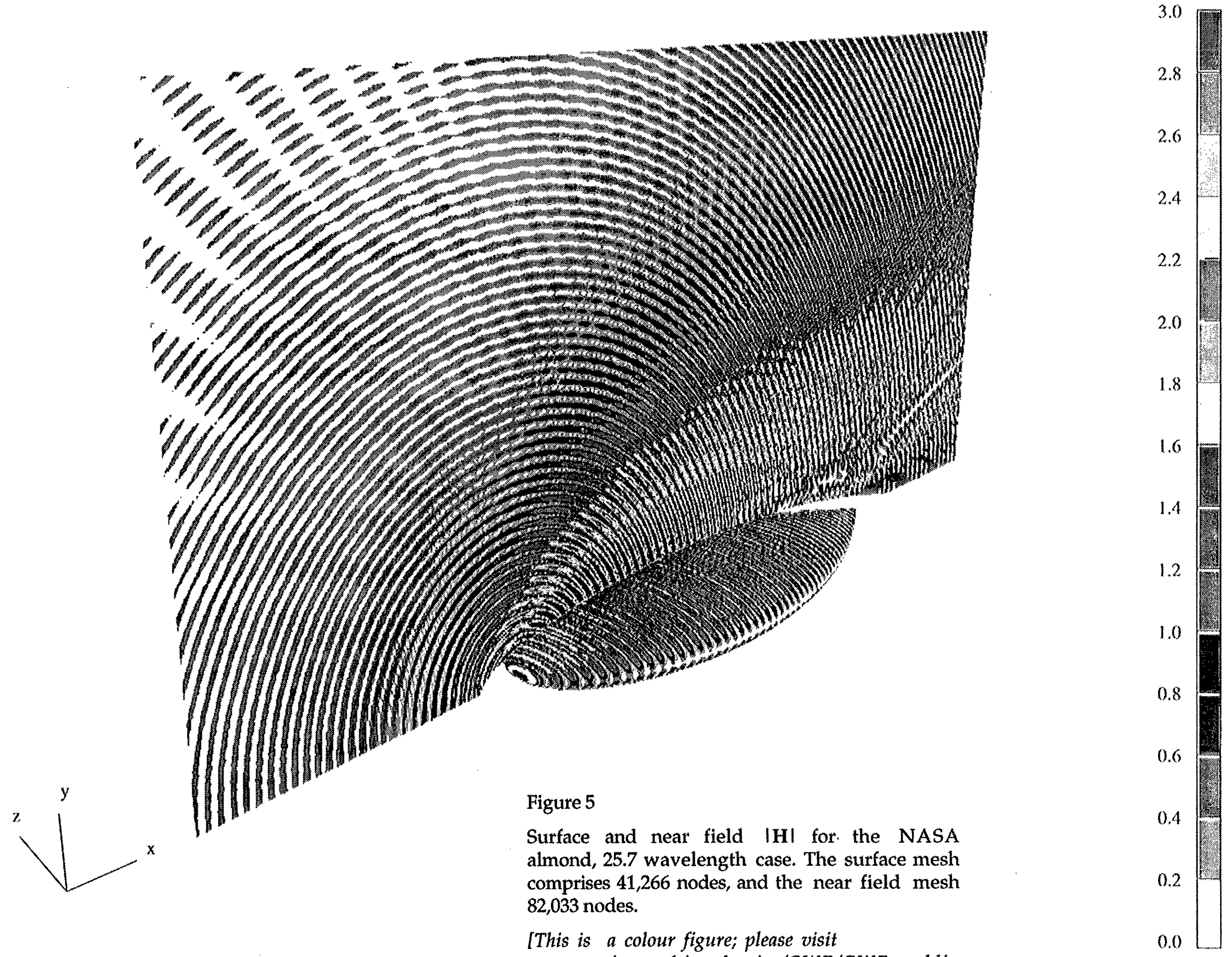


Figure 5

Surface and near field |H| for the NASA almond, 25.7 wavelength case. The surface mesh comprises 41,266 nodes, and the near field mesh 82,033 nodes.

[This is a colour figure; please visit www.me.ic.ac.uk/mechanics/CWP/CWP_publications.html or www.emclab.umr.edu/aces/acesjrnl.htmlto view]

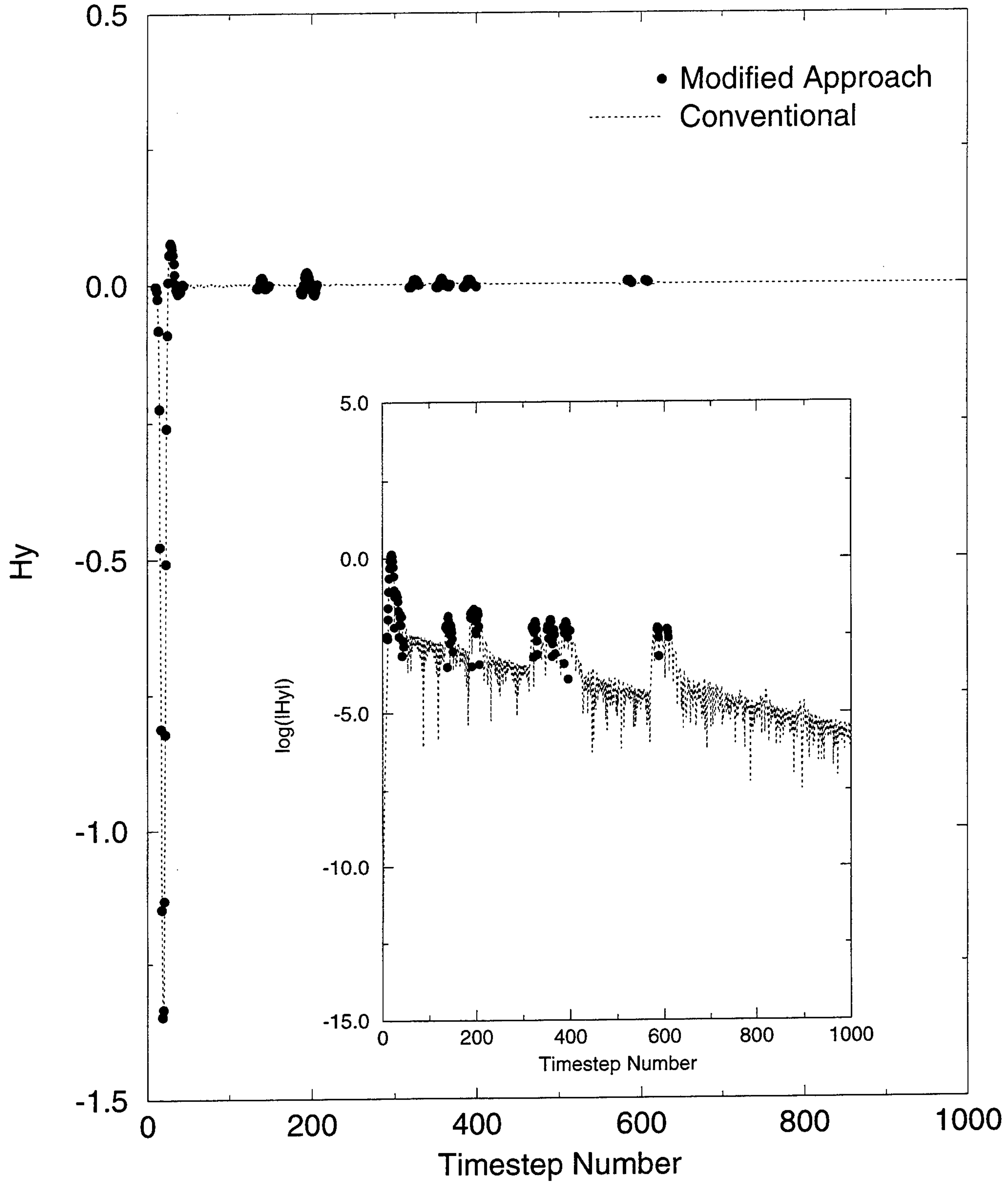


Figure 6

y-component of surface **H** field versus timestep at a node on the array of 10 randomly positioned 98 node spheres, for the conventional and modified approaches. The inset shows this logarithmically.

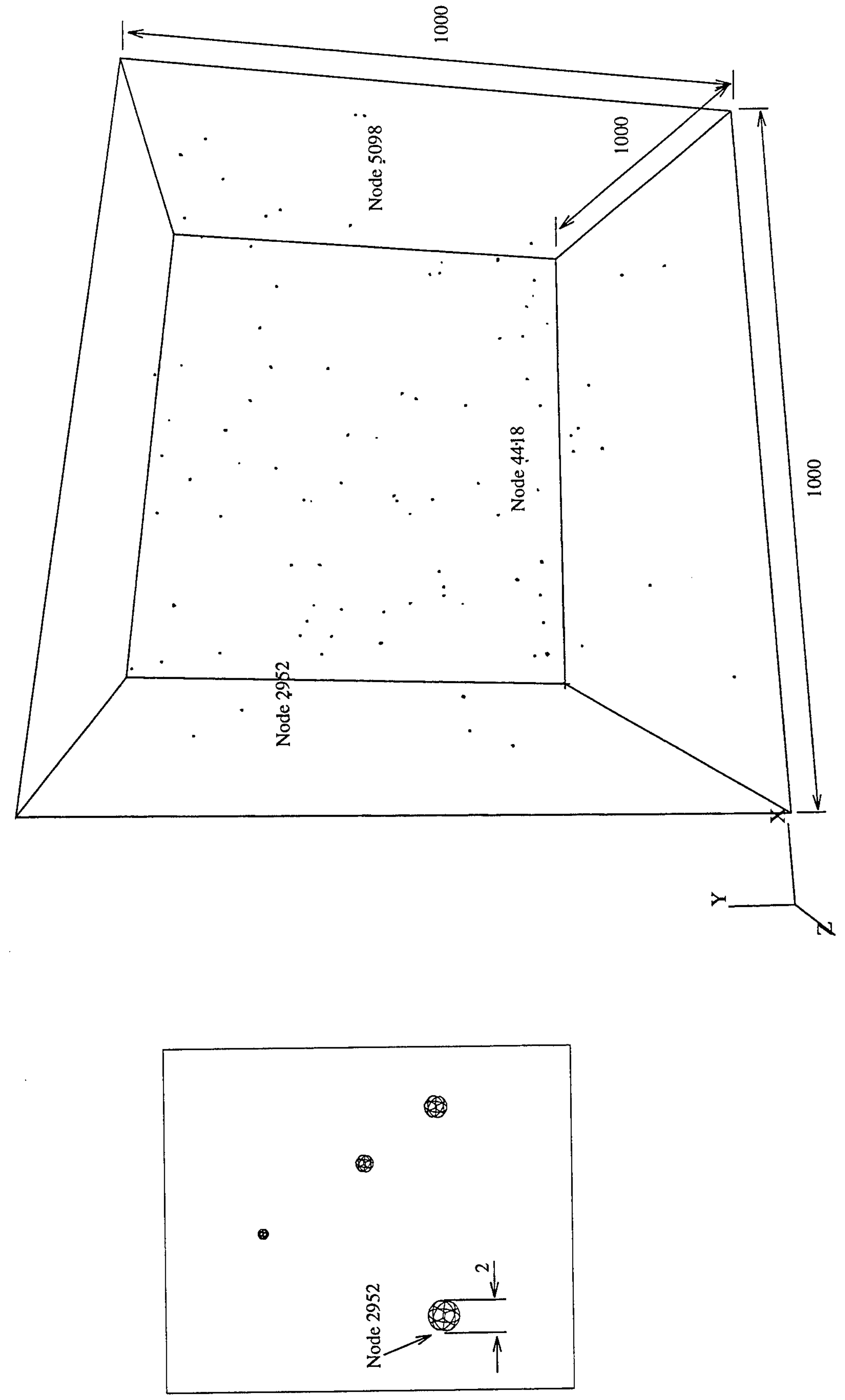


Figure 7

The array of 101 randomly positioned 98 node spheres. The box drawn around the spheres is for illustrative purposes only. The inset shows a close-up of a part of the array.

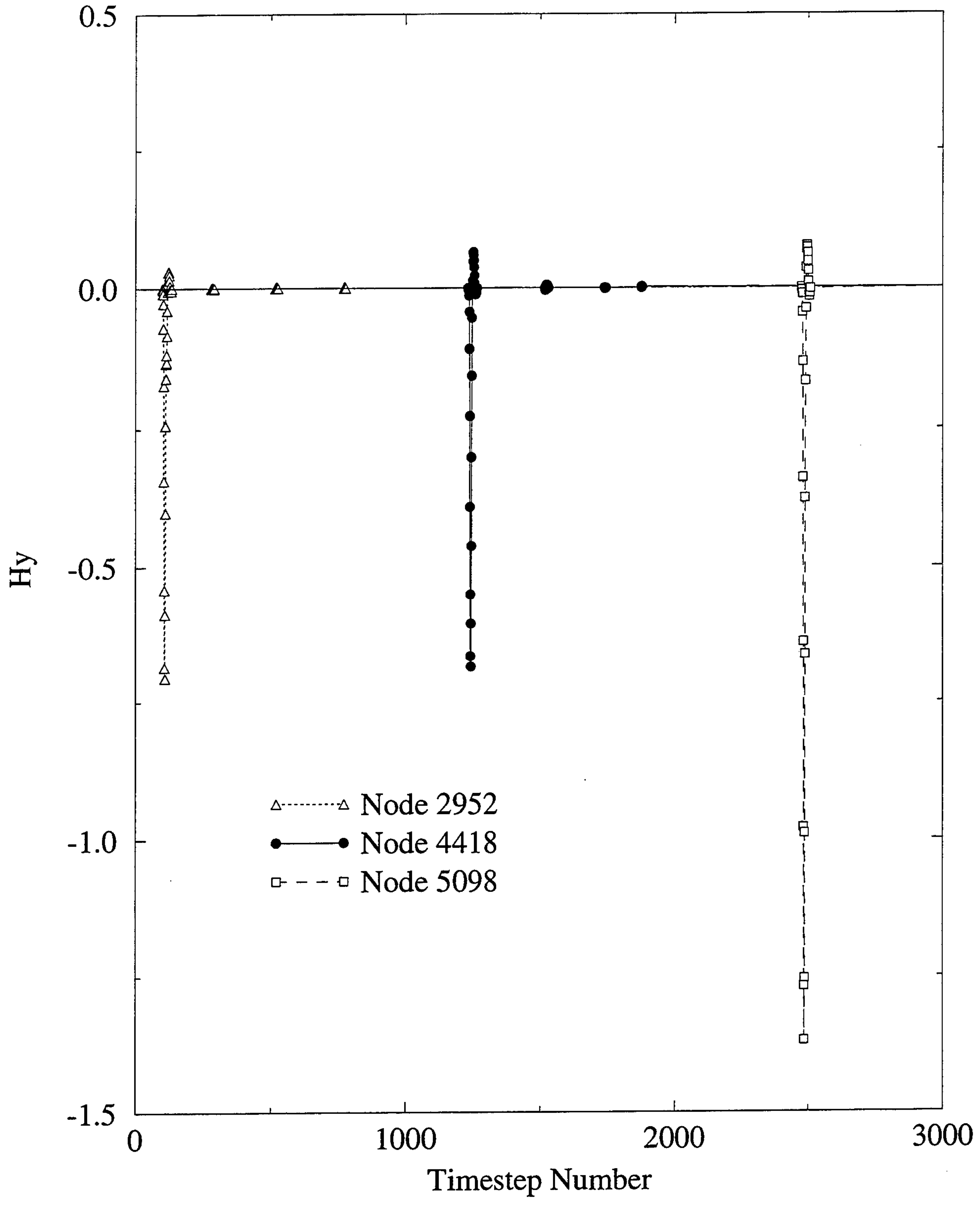


Figure 8

y-component of surface **H** field versus timestep at 3 nodes on the array of 101 randomly positioned spheres shown in figure 7. Propagation is in +x, and **E** polarisation is +z. The nodes' locations are indicated in figure 7.

PERFORMANCE MODELING OF THE FINITE-DIFFERENCE TIME-DOMAIN METHOD ON PARALLEL SYSTEMS*

James E. Lumpp, Jr., Shashi K. Mazumdar,†Stephen D. Gedney
Department of Electrical Engineering
University of Kentucky
Lexington, KY 40506, USA

ABSTRACT. *As high-performance parallel codes are developed or ported to new architectures, it is often difficult to quantify the the causes of performance problems. Models of program performance can provide users with insight into the effect of system and program parameters on performance, can help programmers tune applications, and can help programmers make decisions about processor allocation. This paper introduces a modeling technique applied to the Finite–Difference Time–Domain (FDTD) algorithm. The technique models the performance of an existing application in terms of the size of the problem being solved and the number of processors. The models show that for sufficiently large problem sizes the algorithm performs well. However, for smaller problem sizes or when too many processors are used, the models show that parallel overheads become significant.*

1 INTRODUCTION

Over the last decade, advances in high performance computing have had a significant impact on computational electromagnetics. Specifically, with the decreasing costs of high speed memory, RISC processors, and high speed networks, the size and complexity of practical engineering problems that can be solved using full wave analyses have dramatically increased. Concurrently, the focus of high performance computing has shifted from expensive high-speed single processor computers to relatively low cost multiprocessor computers.

Programming paradigms have changed with the development of parallel systems to optimize the performance of traditional sequential algorithms. Specifically, specialized algorithms have been developed to exploit parallel systems to minimize CPU times and memory usage. Unfortunately, in a parallel computing environment, there are a number of factors that can affect an algorithm's performance that are not obvious to the developer *a priori*. Thus, an invaluable resource for the development of efficient parallel codes is a robust performance analysis tool. Such a utility can enable the programmer to analyze the code and identify inefficiencies for a specific architecture. It can also provide insight needed to fine-tune the algorithm and optimize its performance. Furthermore, for a production level code, a performance analysis tool can provide a close estimation for the optimum number of processors, the optimal decomposition, and the total execution time for a given problem dimension and target parallel system.

Techniques for performance tuning of parallel programs can be broadly categorized into measurement based techniques and modeling based techniques. In measurement based techniques the program is instrumented using hardware or software instrumentation [14, 39, 25, 26]. The instrumented program is executed and information is collected in the form of trace data and this data is analyzed to find performance problems. This process is repeated until the desired level of performance is achieved.

Measurement based techniques suffer from three main disadvantages. First, they require multiple executions of the program on dedicated systems. It can be difficult for programmers to obtain exclusive access to high-performance systems for performance tuning. Second, the instrumentation used during program execution can change the behavior of the program. This can make it difficult to determine how the program would behave without instrumentation. Finally, with measurement based techniques it is difficult to predict the performance if parameters such as the size of the problem, the architecture, or the type of network change.

Modeling based techniques characterize the performance of the program in terms of specific program parameters such as the algorithm, the communication or synchronization behavior, the problem size and system parameters like the number of processors, and network latency. In addition, performance models can be developed for existing and non-existing programs or architectures. Models are useful for comparing the performance of programs on a wide variety of environments as well as different algorithms for a particular program.

In simulation modeling, the salient features of the

*This research was supported by the National Science Foundation under Grants CDA-9502645 and ECS-9624628, the Advanced Research Projects Agency under Grant DAAH04-96-1-0327, and the Army Research Office under Grant DAAH04-94-G-0243.

†Author's current address is: Stratus Computer, Inc., San Jose, CA 95125

program and system are used to drive a simulation [8, 4, 36, 12] . Simulation models offer great flexibility, however, simulations can require orders of magnitude more time to run than the actual program. In addition, accurate simulations require sophisticated and detailed models that can be difficult to generate. For existing programs, direct execution simulation techniques have been developed that greatly improve the simulation time by using the actual programs which has the added benefit of reducing the complexity involved in model creation [36, 4, 31].

Performance models that parameterize program behavior as scalars or mathematical functions can also be developed [29, 10, 38, 13, 1]. These models do not require any time for simulation, however, accurate predictions still require detailed models of the program and computer system[29, 34, 27]. When modeling existing parallel program, the structure of the program can be used as the starting point[3, 9, 7, 33]. Techniques for modeling existing programs typically consist of *static* analysis of source codes to generate the model structure and run-time or *dynamic* analysis to generate the final model.

Several performance modeling tools have been developed that make use of static and dynamic analysis to model parallel programs [3, 9, 7]. The Modeling Kernel [3], which is a part of the AIMS instrumentation toolkit [39], models a program based on the duration of sequential blocks, message lengths and communication phases. APACHE models programs using separate computation models and communication models [7]. APACHE instruments the application to determine the computational requirements, its branching behavior, and the number and size of messages.

In this paper, a performance modeling technique that can help programmers to identify problems in programs or inefficiencies in the interaction between programs and a given architecture is presented. The technique consists of a static modeling process based on the actual program source code and dynamic analysis of the run-time behavior of the program. The dynamic analysis is performed using only the overall execution time of the application. As a result, instrumentation to observe communication patterns or procedures called is not required. This minimizes the intrusion on the run-time behavior of the program.

The performance modeling tool developed is applied to the analysis of the performance of the finite-difference time-domain (FDTD) solution of Maxwell's equations [37, 35]. The FDTD method is based on an explicit time-marching solution and has the advantage that time-variant electromagnetic fields can be accurately and efficiently modeled within inhomogeneous, non-linear, and anisotropic media.

The FDTD algorithm is an excellent algorithm for parallel computer systems. The kernel of the algorithm is a directly addressed sparse matrix-vector multiply and can be efficiently implemented using a simple nested loop structure. Implementation on parallel computers requires message passing only between neighboring processors, and has lead to a highly scalable algorithm [18, 23, 30, 11]. By exploiting parallel architectures based on high-performance RISC processors, the problem sizes that can currently be efficiently solved using the FDTD are orders of magnitudes larger than problems that could have been treated a few years ago.

In this paper the parallel FDTD algorithm is presented. The algorithm is based on the traditional Yee algorithm [37, 40] with a uniaxial PML absorbing media [32, 20, 19]. The performance modeling techniques show that for sufficiently large problem sizes or small numbers of processors, the FDTD algorithm performs quite well. When the problem sizes are small (or number of processors large) the models show that the parallel overheads can become significant and performance decreases. The models also uncovered a performance problem for a particular processor configuration in FDTD. By restructuring loops that communicate boundary information, the performance was substantially improved.

2 THE FDTD ALGORITHM

The finite-difference time-domain algorithm is a direct solution of Maxwell's equations for the electric and magnetic field intensities in a finite, piecewise homogeneous space. The algorithm is based on the discretization of Maxwell's curl equations (Ampère's and Faraday's Laws) using central difference approximations of the spatial and time derivatives. This is achieved by projecting orthogonal components of the vector fields onto the edges of a dual, staggered, orthogonal grid. By staggering the vector fields both in space and time, a second-order accurate explicit time-marching solution is obtained [37].

One of the most challenging aspects of the FDTD method is implementing an absorbing boundary condition that can accurately truncate the mesh over broad frequency bands. The perfectly matched layer (PML) absorbing media introduced by J. P. Berenger [2] has been demonstrated to be a highly effective method for the termination of FDTD lattices [6, 28, 22] and can result in reflection errors as minute as -100 dB. Recently, it has been shown that the PML method can be reposed in a Maxwellian form as a uniaxial anisotropic medium [32, 20, 19]. It has also been demonstrated that the uniaxial medium can be perfectly matched to a lossy, inhomogeneous, dispersive, isotropic and anisotropic medium [20]. Most significant is that the extension to such complex media in a FDTD implementation is quite trivial.

The time-dependent electric and magnetic fields within the uniaxial PML are computed using an explicit time-marching solution scheme, as derived in [20, 19].[1] The uniaxial PML can be easily and efficiently implemented within the framework of an existing FDTD code. For example, posing a uniaxial PML throughout the entire space, the discrete field updates can be expressed as a triple-nested loop (illustrated here in FORTRAN):

```
  do 10 k = 1,nz-1
    do 10 j = 2,ny-1
      do 10 i = 2,nx-1
        ds = dz(i,j,k)
        dz(i,j,k) = ay(j)*dz(i,j,k) +
                    by(j)*[hy(i,j,k)-hy(i-1,j,k)-
                    hx(i,j,k)+hx(i,j-1,k)]
        ez(i,j,k) = ax(i)*ez(i,j,k) +
                    bx(i)*[az(k)*dz(i,j,k)-bz(k)*ds]*
                    er_z(i,j,k)
  10 continue
```

where

$ay(j) = \frac{2\varepsilon_0\kappa_{y_j}-\sigma_{y_j}\Delta t}{2\varepsilon_0\kappa_{y_j}+\sigma_{y_j}\Delta t}$, $by(j) = \frac{2\varepsilon_0\Delta t}{2\varepsilon_0\kappa_{y_j}+\sigma_{y_j}\Delta t}\cdot\frac{\Delta z}{\Delta x\Delta y}$,

$ax(i) = \frac{2\varepsilon_0\kappa_{x_i}-\sigma_{x_i}\Delta t}{2\varepsilon_0\kappa_{x_i}+\sigma_{x_i}\Delta t}$, $bx(i) = \frac{2\varepsilon_0\Delta t}{2\varepsilon_0\kappa_{x_i}+\sigma_{x_i}\Delta t}$,

$az(k) = \frac{\kappa_{z_k}}{\Delta t} + \frac{\sigma_{z_k}}{2\varepsilon_0}$, $bz(k) = \frac{\kappa_{z_k}}{\Delta t} - \frac{\sigma_{z_k}}{2\varepsilon_0}$,

$er_z(i,j,k) = \frac{1}{\varepsilon_0\varepsilon_r(i,j,k)}$

and the fields have been scaled by their edge length (e.g., $E_x = \Delta x E_x$). It is noted that the PML parameters σ_i and κ_i $(i = x, y, z)$ are one-dimensional variables. Specifically, in the interior working volume, it is assumed that $\sigma_i = 0$, and $\kappa_i = 1$, and in the PML regions, they are assumed to have an m-th order polynomial spatial variation along their respective axes. As a result, the update coefficients above are simply one-dimensional coefficients.

Updating the fields over all space has the limitation that the additional storage arrays required to store the flux densities (e.g., Dz) must be stored over all space. However, it does offer the advantage of simplicity in the modification of existing codes. An alternative is to write a triple nested loop for the interior fields, and then write separate loops for the different PML regions (segregating corner regions). Then, only the auxiliary variables need to be stored in the PML regions, leading to memory savings. In this circumstance, the uniaxial PML will require considerably less storage than Berenger's PML, because only the normal fields require dual storage as opposed to the two tangential fields as required by Berenger's PML formulation. Based on this scheme, the FDTD with a uniaxial PML truncation on all 6 boundaries will require

$$6N_xN_yN_z + 8N_{pml}(N_xN_y + N_yN_z + N_zN_x) -$$
$$16N_{pml}(N_x + N_y + N_z) + 24N_{pml}^2$$

real numbers as compared to $12N_xN_yN_z$ real numbers required by a FDTD method with PML everywhere [21].

In this paper, the UPML is assumed to be distributed throughout the grid. While this increases the memory overhead, the overall computational time is slightly better than the case when the grid is split up into multiple regions [21]. Furthermore, load balancing is readily achieved.

3 PARALLEL IMPLEMENTATION OF THE FDTD ALGORITHM

It has been demonstrated that the finite-difference time-domain algorithm is well suited for implementation on tightly coupled distributed memory high performance parallel computers [30, 5, 15, 18]. This is principally due to the regularity of the dual grid and the even distribution of effort throughout the grid during the entire time-marching solution. Furthermore, only the magnetic fields tangential to the shared boundaries needed to be communicated between processors each time iteration [18]. The algorithm presented in this section is primarily focused on distributed memory multicomputers with a single program multiple data (SPMD) paradigm.

The parallel algorithm is based on a spatial decomposition of the regular, orthogonal FDTD lattice. To this end, the original domain is spatially decomposed into contiguous sub-domains. The sub-domains are rectangular in shape, non-overlapping, sharing common surfaces only, and are of equal size. The boundaries, or surfaces, shared by sub-domains are chosen by taking slices along edges of the primary grid along the x, y, and z-directions. Each sub domain is then mapped directly onto independent processors of the parallel computer.

Assume that the lattice has dimension $N_x \times N_y \times N_z$. The lattice is then discretized using a three-way dissection. Essentially, assume that the lattice is mapped onto a three-dimensional grid of processors (P_x, P_y, P_z), where P_x, P_y, and P_z are the number of subsections along the x, y, and z-directions, respectively. (Also, the total number of processors will be $P = P_xP_yP_z$). Then, given a global grid with dimensions N_x, N_y, and N_z, each processor will be assigned a block of the grid with dimensions $N_x/P_x, N_y/P_y$, and N_z/P_z . For most applications, these ratios will be non-integer values, and the sizes of the grids will be slightly uneven, resulting in some load imbalance.

The local grid dimension for each processor, given as $nxp \times nyp \times nzp$, can be uniquely determined using a simple algorithm. Each processor is first assigned a coordinate (p_x, p_y, p_z), where $p_x \in (1, P_x), p_y \in (1, P_y)$, and $p_z \in (1, P_z)$. Then, each processor can uniquely determine its grid dimension *nxp* along the x-direction using the algorithm

[1]The readers are referred to these references for the general theoretical development of the uniaxial PML.

```
$nxp = aint(N_x/P_x)$
$if(mod(N_x, P_x).le.p_x)nxp = nxp + 1$
```

where, aint() is the FORTRAN intrinsic function which truncates the argument to an integer, and mod() is the modulus FORTRAN intrinsic function which computes the remainder of the quotient N_x/P_x. This algorithm assures that no processor has more than one additional row of the grid than any other processor. The dimensions *nyp* and *nzp* can be computed in a similar manner. The parallel algorithm will then consist of updating all fields assigned to each processor independently, with special consideration for discrete field components that lie on the boundary interfaces shared by two sub-domains.

The spatial decomposition is chosen to slice along the primary lattice grid faces. Specifically, in the boundaries shared between any two processors, the discrete electric field vectors in the planar boundary are tangential to the surface. Thus, the discrete magnetic field vector is normal to the boundary interface. The fields on the shared boundary interface are redundantly stored in memory on both processors sharing that boundary. Due to the decomposition described, multiple processors can share a lattice edge. The discrete electric field vectors associated with the edge are assumed to be stored redundantly on all processors sharing the edge.

The magnetic field vectors normal to the shared interface can be updated independently on all processors sharing the face. The update is proportional to the line integral of the electric field about the edges bounding the face. Because each processor has the updated value of the tangential electric fields on the shared interface, the normal magnetic field can be updated independently on each processor. Therefore, interprocessor communication is not needed when updating the magnetic fields within each sub-domain. Rather, it is much more expedient to simply update the normal magnetic fields in the shared boundary redundantly on each processor sharing the face.

On the other hand, the discrete tangential electric field vectors on the shared boundary interface do not have enough information locally to perform the update. For example, consider an edge shared by two processors. Three of the four magnetic field vectors needed for the update are stored in local memory. The fourth magnetic field vector, which is tangential to the interface, is in memory on the adjacent processor. Hence, this data must be communicated to the local processor before the update can be completed.

The parallel algorithm performing the parallel FDTD algorithm is illustrated in Figure 1. The first step is to update the magnetic field intensity in all space. The next step is to send the magnetic field vectors tangential to the shared boundary (one-half cell removed) to the neighboring processor, while receiving the complimentary magnetic fields across the shared face and storing them in a local vector. To reduce the effects of message passing overhead, all the discrete magnetic fields to be communicated to a given processor are first combined into a single buffer and then sent to the adjacent processor. The electric fields are then updated, including those on the shared boundary. Note that the electric fields on the shared boundaries are updated redundantly on each processor to avoid additional communication.

```
initialize e,h to zero
do it = 1, max_iterations
        call source_update
        call h_update
        call communicate_h_field
        call e_update
enddo
```

Figure 1: Parallel FDTD Algorithm

It will be shown that the algorithm described is quite scalable. However, because the interprocessor communication must be performed at each time step, the efficiency of the parallel algorithm will ultimately stagnate as the number of processors is increased for a fixed problem size, as predicted by Amdahl's law. It will be shown that the performance analysis tool described in the following section can predict the optimum number of processors for a given problem size.

4 MODELING TECHNIQUE

Our approach to model the performance of parallel programs is by analyzing the structure of programs and then measuring the performance from actual program runs as key factors of the application and architecture are varied. This information is used to create a closed-form expression for the execution time of the program in terms of those factors.

The performance of a parallel program depends on many different attributes from architecture details to the structure of the algorithm. Some of these factors are under the control of the user and can be changed to improve performance. However, there are other factors that either can not be changed or the changes cannot be controlled by the user. While modeling techniques similar to those presented here can be used to model other factors affecting program performance (e.g., processor speed), the goal of the modeling technique presented here is to assist users in modeling programs primarily in terms of factors that can be controlled by an end user of a given parallel machine.

Examples of factors that are constant or change very rarely include the speed of the processors, the type of interconnection network or its topology, or the size and speed of memory. While these factors directly influence performance, they can not be changed in a typical end-user environment. These factors are assumed to be constant and are included implicitly in our models.

Factors affecting performance that can change, but may not be under the control of the user, include the external load on individual machines or external load on the network (which can impact latency and available bandwidth). If the parallel machine is dedicated to a single user, these factors will not be a concern. However, for a typical distributed system these factors can greatly impact performance. These factors are implicitly included in the models if the dynamic analysis can be done with these factors at values similar to those that will be present when the program is used.

Typical factors that the user can control include the size of the problem, the number of processors used, and the algorithm used. The modeling approach presented here analyzes the program structure in terms of the control flow, loop structures and the communication and synchronization events.

During the static analysis phase, the program source is analyzed to generate a model template for the execution time. The model is based on identifying the basic blocks, loops, function calls, communication, and synchronization events. By determining the bounds on loops and the number and size of messages in terms of the problem size (S) and the number of processes (P), it is possible to determine the terms that will appear in the execution time model. A static model for execution time in terms of S and P is generated.

The final model of execution time is found by determining the coefficients of the terms in the static model. These unknown coefficients are estimated using regression on execution time data gathered during run-time. The following sections describe the approach in detail.

4.1 Static Analysis

The execution time of a parallel program can be divided into three categories. These are:

- **Computation Time:** This is the portion of time spent in performing actual computation.
- **Parallel Overhead Time:** This is the portion of time spent in exchanging information between the processes.
- **Synchronization Time:** This is the amount of time that the processes spend in coordinating their activities.

The computation time is the time spent performing actual computations that are required in the program. It depends on the number of operations performed in the program, and is often referred to as the *order* or *complexity* of the computation. The computational complexity of a program depends on the number of loops and routines that are present.

In a parallel program, there is always some overhead associated with transferring information between processes. The amount of time it takes to format and prepare to send a message or to initiate an access to a remote memory location is defined as the *Parallel Overhead Time.* This measures only the local time to process and send a message or to make a remote memory access and not the time it takes for messages to propagate between processes because the transmission time may overlap with computation or parallel overhead time.

Some form of synchronization between the processes is required to coordinate the activities in a parallel program. This is usually achieved by implementing locks, semaphores or barriers in shared memory systems or synchronous (blocking) communication calls in message passing systems. The time spent by a process blocked on a synchronization event is defined as *synchronization time.*

To accurately model the execution time of a parallel program, each of the three categories must be modeled. The following sections explain the techniques used to model each of these.

4.1.1 Computation Time

Computation time models are generated by dividing the program into *basic blocks* which are the largest consecutive blocks of instructions between control flow constructs in the program. The structure of each loop in the program is analyzed to determine the number of times loops will iterate to find the number of times each basic block will execute. If it is assumed that the execution time of a basic block is constant, then the execution time for a loop is the execution time of the basic block multiplied by the number of times the loop iterates.

For example, if the execution time for a given basic block in a loop is k then the total time for the basic block for n iterations of the loop would be $k*n$. For many loops it is possible to statically determine the number of iterations by analyzing the upper and lower bounds of the loop. Consider a loop l with an upper bound of u, a lower bound b, and the stride through the loop is s, then the number of iterations of the loop can be expressed as:

$$I_l = \frac{u-b}{s}$$

and the execution time for loop l can be expressed as:

$$E_l = k * \frac{u-b}{s} = k * I_l$$

If u, b and s are expressed in terms of the problem size S and the number of processors P, the execution time can then be expressed as:

$$E_l(S,P) = k * I_l(S,P)$$

Nested loop structures in programs must also be modeled. Consider a loop g with $I_g(S,P)$ iterations that is nested within a loop l. For each execution of the outer loop, the inner loop iterates $I_g(S,P)$ times. Thus, the total execution time of the outer loop can be written as:

$$E_l(S,P) = I_l(S,P)(k_0 + k_1 * I_g(S,P))$$

This can be generalized to any number of loops within l as follows:

$$E_l(S,P) = I_l(S,P)(k_0 + \sum_n k_n * I_n(S,P)$$

where $I_n(S,P)$ is the number of iterations of the nth nested loop. It is also possible for procedure calls to be nested within a loop. If the execution time of a routine i is R_i, the overall execution time of a loop, including all the nested loops and the procedure calls, is then be expressed as:

$$E_l(S,P) = I_l(S,P)(k_0 + \sum_n k_n * I_n + \sum_i R_i(S,P))$$

where $R_i(S,P)$ is the execution time of the ith procedure call within the loop.

The execution time of a routine or a procedure is the sum of the execution times of all the loops present, as well as the execution times of all other calls made within the scope of the particular routine. The execution time of the routine can then be expressed as:

$$R_i(S,P) = \sum E_j(S,P) + \sum R_k(S,P)$$

where E_j is the execution time of the jth loop and R_k is the execution time of the kth called function within the routine. The overall execution time for any application then becomes the execution time of the main routine.

4.1.2 Parallel Overhead Time

Parallel programs must exchange information throughout program execution. The time to manage and initiate transfers can constitute a significant portion of the execution time for a program. The overhead time depends on the number of messages sent and the amount of information passed.

Communication time consists of two parts, a fixed overhead and a variable overhead portion. The fixed overhead is the time that a process takes to initialize message buffers, set up a network connection with the receiver, etc. The variable overhead is proportional to the size of the message, for example, the time to copy messages buffers and the time it takes to propagate the message to the network. In our models, the constant overhead can be attributed to the basic block in which the communication call is made. Therefore, the only additional term necessary in the model is proportional to the size of the message. These terms are added to the computation time model before dynamic analysis.

4.1.3 Synchronization Time

In a parallel program, processors must synchronize to coordinate their activities. It is not possible to model the synchronization time through static analysis of the program, because synchronization time depends not only on the problem size S and the number processors P, but also on non-deterministic factors like the load on each processor and the contention in the network.

Synchronization in message passing parallel programs is achieved with barriers or synchronous communication calls. In a barrier, all processes block at a particular point during their execution and wait for all other processors to block. Barriers and global reductions, such as summing vectors, are implemented with $\log P$ algorithms. To model this type of behavior, $\log P$ and $P * \log P$ terms are included in the static models.

4.1.4 Static Analysis Tool

A tool to automatically generate the static model described in the previous section was developed. The tool, based on the Sage++ parsing toolkit [17], parses the program to identify all routines, control flow constructs, communication, and synchronization calls. Each loop in the program is identified and the associated execution time model is determined based on the loop bounds. For loops where it is not possible to statically identify the number of iterations (e.g., a loop bound that depends on input data), the user is prompted for the value if the user does not know the number of iterations, a heuristic is used where the loop number of iterations is assumed to match the number of iterations of some other loop in the program. The calculation of overhead times for messages is also done automatically by analyzing the communication calls. If the size of the message can not be determined statically, again a heuristic can be employed or the user can supply the expected size. The tool can determine the execution time of any particular routine. Analysis of the main routine provides a model for the overall execution time.

4.2 Dynamic Analysis

The unknown coefficients $(k_0, k_1, \cdots)$ in the static model are determined by dynamic analysis to produce the final model for execution time in terms of S and P. Consider the equation

$$E_{time} = k_0 + k_1 * (f_1) + k_2 * (f_2) + \cdots + k_n * (f_n)$$

where, f_n's are polynomials of (S, P) and $K = (k_0, k_1, \cdots, k_n)$ are the unknown coefficients. Let this equation be the execution time model for a hypothetical parallel program. This equation has $n + 1$ terms including a constant term. To determine the coefficients for this expression, it is necessary to have at least $n + 1$ equations. Hence, at least $n + 1$ runs of the program must be performed to generate the final expression. The system of equations can be represented in a matrix form as follows:

$$F(S, P) * K_n = E_n$$

where $K_n = (k_0 k_1 k_2 \cdots k_n)^T$ and $E_n = (e_0 e_1 e_2 \cdots e_n)^T$. E_n represents execution times for unique values of (S, P). The coefficient vector K_n can be determined from $F(S, P)$ and E_n using:

$$K_n = F(S, P)^{-1} * E_n.$$

All possible combinations of the terms present in the static model are checked to determine which set of factors produce the best model. The measure for each model is the *coefficient of determination* (R^2) which is defined as the fraction of the variation that is explained by the model[27].

We have developed a dynamic analysis tool that automates the process of model generation. The *Dynamic Analysis Tool* generates models for all possible combinations of the terms using least squares curve fitting. The coefficient of determination for each of the model is calculated and compared. Plots of execution time versus problem size or the number of processors can be generated allowing the user to visually compare the models. Plots of percentage error versus the problem size of the number of processors are also generated.

5 PERFORMANCE MODELING OF THE FDTD PROGRAM

The Finite Difference Time Domain (FDTD) program is implemented in Fortran using the Parallel Virtual Machine (PVM) message passing library [24], however the same analysis would result if another message passing systems was used, e.g., MPT[16]. FDTD was modeled on two platforms typical of current high-performance architectures: an SGI Power Challenge and a cluster of Sun Hyper-Sparc workstations on a 100 Mbps ethernet.

The FDTD program consists of approximately 158 loops (this includes all the initialization loops, field update loops, and loops to support interprocessor communication). The majority of the computation takes place within the triply nested field update loops. However, it was found that the doubly nested loops used for interprocessor communication can be non-negligible. As explained in section 2, the algorithm is based on a spatial decomposition of the regular, orthogonal FDTD lattice of dimensions N_x, N_y and N_z. This lattice was then mapped onto a three dimensional grid of processors (P_x, P_y, P_z) to allocate the work to individual processors. The problem size of the algorithm can be changed by altering the dimensions of the lattice. It is also possible to change the configuration of the processors in the grid and thereby changing the manner in which the workload is distributed. As an example, Figure 2 shows different processor configurations possible when using 8 processors. In this paper, it will be assumed that $N_x = N_y$ and N_z is constant. It will also be assumed that $P_z = 1$. Subsequently, the grid of processors will be two dimensional (i.e., $(P_x, P_y, 1)$). This is typical of microwave circuit analysis, where $N_z << N_x, N_y$ [18]. While the performance analysis tool is not restricted to this, it will greatly simplify the presentation of data.

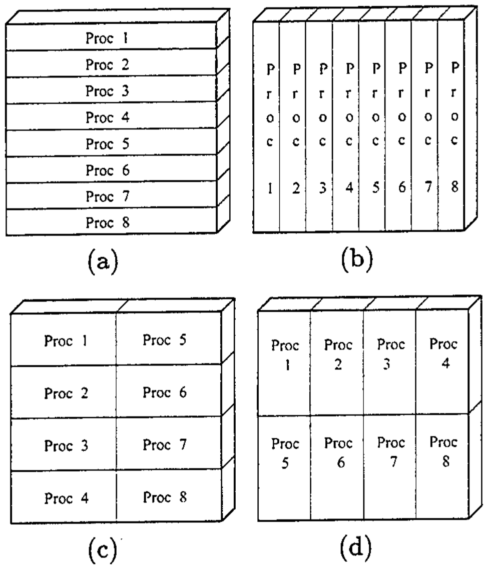


Figure 2: Configurations for Eight Processors; (a) (1,8,1), (b) (8,1,1), (c) (2,4,1), (d) (4,2,1).

The static analysis tool described in section 4.1.4 was first used to model the FDTD algorithm. Under the constraints defined above ($N_x = N_y$ and N_z = a constant), the problem size S was set to N_x. The model was then found to be:

$$E_{time} = k_0 + k_1 * P + k_2 * N_x + k_3 * N_x/P$$
$$+ k_4 * N_x^2/P + k_5 * log(P) + k_6 * P * log(P)$$

where P is the number of processors ($P = P_x P_y P_z$). This model includes terms proportional to the size of the problem $k_2 * N_x$, the number of processors $k_1 * P$. There are also two terms that show the computation time proportional to the problem size divided by the number of processors $k_3 * N_x/P$ and the problem size squared divided by the number of processors $k_4 * (N_x^2/P)$. These terms correspond to the computation being divided among the processors. It is anticipated that the squared term will dominate, since for a fixed N_z, the field update computations will be proportional to $N_x * N_y = N_x^2$.

Each communication in the program is modeled by a constant term and a term proportional to the message size. Because each communication call is made within some basic block, and each basic block is modeled with a constant term k_i, the constant term of the communication call is included in the model of the basic block. To include the term proportional to the message size, the communication call is modeled as a call to a routine with the appropriate execution time. The size of the message is determined and expressed in terms of N_x and P to generate the model for communication. As described in section 4.1.3 the static analysis tool also included the terms $k_5 * \log P$ and $k_6 * P * \log P$ typical of synchronization behavior in parallel programs.

Models in terms of the problem size or the number of processors alone can also be generated by simplifying the overall model. The model in terms of problem size is:

$$E_{time} = k_0 + k_1 * N_x + k_2 * N_x^2$$

and the model based on the number of processors is:

$$E_{time} = k_0 + k_1 * P + k_2 * 1/P + k_3 * log(P) + k_4 * P * log(P).$$

Dynamic analysis experiments were run on both the SGI Power Challenge and the Sun network. On both the platforms PVM 3.3.11 was used for message passing. On the SGI, which is a shared memory machine, the PVM shared memory port was used so that messages were efficiently routed through shared memory. In the cases illustrated, run-time information was collected for different problem sizes varying N_x and N_y from 31 through 131 and the number of processors was varied from 2 to 8. In all cases, $N_z = 21$, and $N_x = N_y$. The static and dynamic models include both the set up time and the explicit field updates. The simulations were run for 1000 time iterations.

While experimenting with 8 processors, models were also generated for all the different possible processor configurations namely (1,8,1), (8,1,1), (2,4,1), and (4,2,1). The one-way dissections (i.e., (1,8,1) or (8,1,1)) require interprocessor communication in one-direction only. Whereas, two-way dissections (2,4,1) or (4,2,1)) require communication in two-directions. This poses a trade off in the sense that the amount of data communicated may be reduced; however, the number of interprocessor communications per processor is increased. The loop dimensions also vary with the geometry of the decomposition which will also have an affect on the code's performance.

Initially, models in terms of problem size were generated by keeping the number of processors constant at 4, 6 and 8. Based on the static analysis, and a dynamic analysis for small problem sizes ($N_x = N_y = 51$ and $N_x = N_y = 61$), models for the SGI Power were derived based on the formulations in Section 5. Both models were based on a simulation with 1000 time iterations. Table 1 shows the models for P = 4, 6, and 8.

# of processors	k_0	k_1	k_2
4	6.257	0.000	0.021
6	-1.067	0.000	0.016
8	2.845	0.000	0.011

Table 1: Coefficients for the model $E_{time}(N_x) = k_0 + k_1 * N_x + k_2 * N_x^2$ on the SGI Power Challenge.

This analysis was repeated for the 100 Mbps network of Sun Hyper-Sparcs. Table 2 shows the size models on the network of Hyper-Sparcs for P = 4,6, and 8.

# of processors	k_0	k_1	k_2
4	51.050	0.000	0.113
6	82.515	0.000	0.097
8	47.069	0.000	0.083

Table 2: Coefficients for the model $E_{time}(N_x) = k_0 + k_1 * N_x + k_2 * N_x^2$ on the Sun Hyper-Sparcs.

Figures 3 and 4 show the models on the SGI Power Challenge and the Sun workstations respectively. From the models it can be seen that the execution time varies as square of the problem size. This can be attributed to the fact that the program has a significant amount of computation that is performed within the triply nested loops with loop bounds that depend on N_x, N_y and N_z, where N_z is a constant.

From Figure 3, it can be seen that the predicted times matched closely with the actual execution times. The original models were generated using small values of problem sizes such as $N_x = N_y = 51$ and $N_x = N_y = 61$. The models were then used to compute the anticipated execution time as N_x and N_y were varied from 1 to 200. This is compared to actual execution times recorded for various values of N_x and N_y. The errors in predictions ranged from about 5 to 8 percent. Similarly, from Figure 4, the error in the predictions for the network of Sun

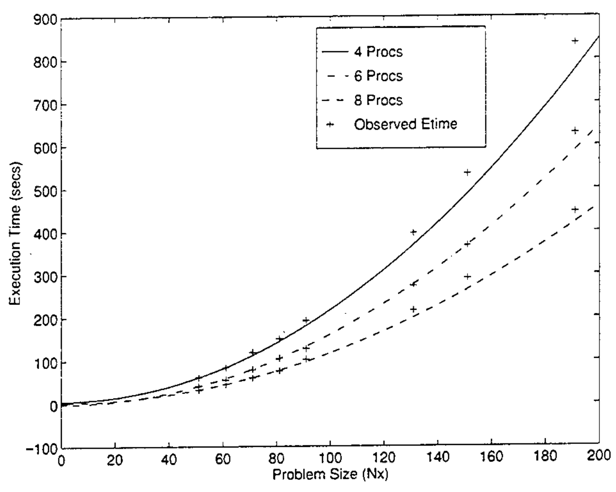


Figure 3: Problem Size Models on SGI Power Challenge, where $N_y = N_x$, $N_z = 21$ (1000 time iterations).

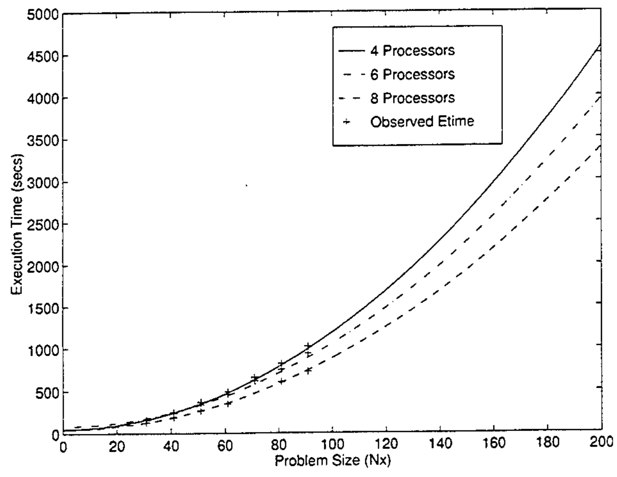


Figure 4: Problem Size Models on Sun Hyper-Sparcs, where $N_y = N_x$, $N_z = 21$ (1000 time iterations).

Hyper-Sparcs was slightly higher than on the SGI and ranged from 6 to 10 percent.

It can also be observed from the models that the performance of the SGI Power Challenge is an order of magnitude greater than the network of workstations. This is because the MIPS R10000 processors in the SGI Power Challenge machine are much faster than the Sun Hyper-Sparcs. Runs with a single processor configuration on each architecture showed that the MIPS R10000 processor to be approximately 8 times faster that the Sun Hyper-Sparc processor.

On the SGI Power Challenge, processors configurations of (1,8,1), (8,1,1), (2,4,1), (4,2,1) were also modeled and compared. Table 3 shows the models for the different configurations. Figure 5 shows the models for the four configurations. It can be seen that as the problem size increases, the (8,1,1) and (4,2,1) configurations perform significantly worse than the (1,8,1) and (2,4,1) configurations.

Configuration	k_0	k_1	k_2
(1,8,1)	2.845	0.000	0.011
(8,1,1)	-7.947	1.037	0.013
(2,4,1)	2.918	0.000	0.012
(4,2,1)	-1.290	0.000	0.016

Table 3: Coefficients for the model $E_{time}(N_x) = k_0 + k_1 * N_x + k_2 * N_x^2$ on the SGI Power Challenge.

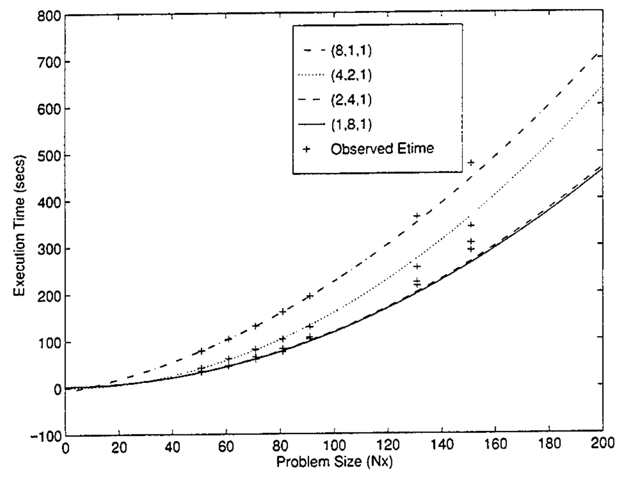


Figure 5: Models for different processor configurations on the SGI Power Challenge, where $N_y = N_x$, $N_z = 21$ (1000 time iterations).

To quantify this behavior additional models were created. Models for the message size and the number of messages showed that the number and size of messages being passed were the same for both the (8,1,1) and (1,8,1) configurations. Thus, the problem was not communication overhead. The next hypothesis was that there was a problem in the computation phase of the program. Because the amount of work done in both the configurations should be equal, interaction with the memory system was suspected. The number of page faults generated by each configuration was measured and is shown in Figure 6 versus the problem size for the (1,8,1) and (8,1,1) configurations. The figure shows that the number of page faults for the (8,1,1) configuration is much higher. The likely cause for page fault problems is irregular accesses to memory.

Further examination of the code revealed that the cause was a set of loops in the interprocessor communication subroutine that accessed a matrix row-wise fashion. In Fortran, row-wise accesses cause non-unit strides through memory that can increase the number of pages touched and, as a result, the number of page faults. The code was modified by changing the loop structure so that one instance of the non-unit stride in the program was removed. Figure 7 shows the performance models for the original version of the (1,8,1) and (8,1,1) configurations and the optimized version of the (8,1,1) configuration. The figure shows that there was a significant improvement in the performance of the modified version when compared to the original program. Although the number of operations performed in the double-nested loops was insignificant compared to the three-dimensional field update loops, it had a dramatic affect on the overall program performance. This illustrates how performance models can be used to uncover unexpected behavior in applications.

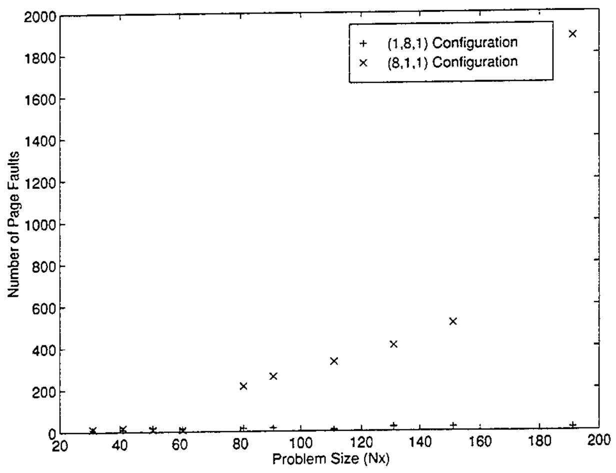


Figure 6: Number of page faults on the SGI Power Challenge for two configurations.

Models in terms of the number of processors were also generated on both platforms. These models were created by changing the number of processors while keeping the problem size fixed. Table 4 shows the different P models for $N_x, N_y = 31$, $N_x, N_y = 81$, and $N_x, N_y = 151$.

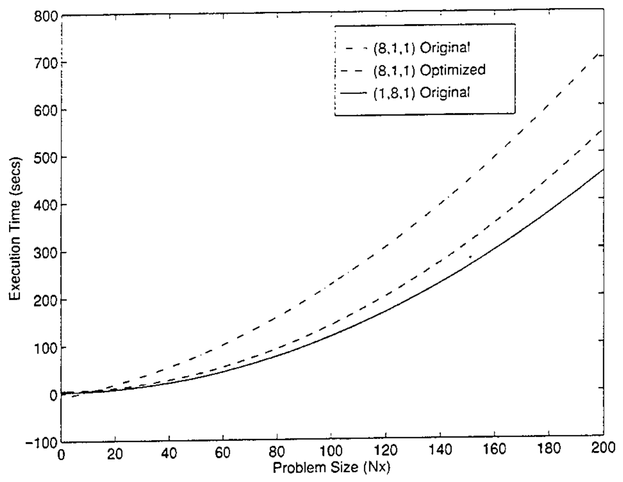


Figure 7: Comparison of original program to the modified version.

Size	k_0	k_1	k_2	k_3	k_4
31	73.081	8.951	0.000	63.704	0.000
81	-106.964	0.000	780.309	44.899	0.000
151	2.945	0.000	2122.639	0.000	0.000

Table 4: Coefficients for the model $E_{time}(P) = k_0 + k_1 * P + k_2 * 1/P + k_3 * \log P + k_4 * P * \log P$ on the SGI Power Challenge.

The different models are shown in Figure 8. From the figure it can be seen that as the number of processors increase the performance, in terms of execution time, improves. As the number of processors increase there is a decrease in the performance gain. This is expected since as the number of processors increase the parallel overhead also increases and this affects the performance. From the figure it can be seen that for problem size of $N_x = N_y = 31$ and $N_z = 21$, the execution time drops gradually until 9 processors and as more processors are increased the execution time increases. This is because at this point the processors are spending more time exchanging information than performing the computation. The optimum number of processors can be determined from the models.

Figure 9 shows the models for the execution time in terms of the number of processors on the network of Sun Hyper-Sparcs. The models were generated for $N_x = N_y = 31$, $N_z = 21$ and $N_x = N_y = 91$, $N_z = 21$. Similar to the models on the SGI Power Challenge, the performance of the program improves sharply at first and then gradually reduces as the number of processors are increased. The coefficients for the models are shown in Table 5.

Performance models for the FDTD application in terms of both the problem size and the number of processors were also generated. The model for the execution

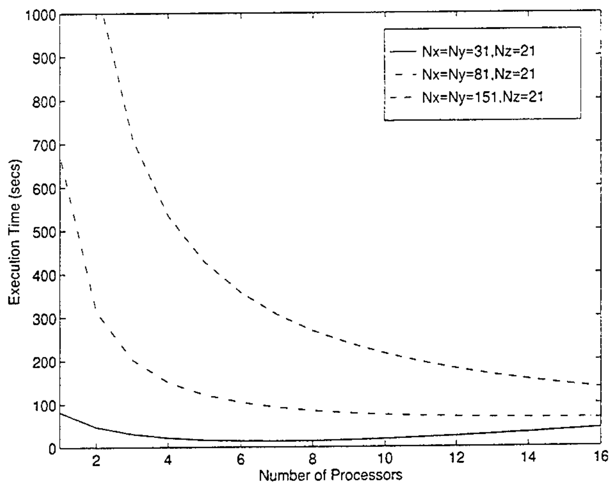


Figure 8: Processor Models on the SGI Power Challenge (1000 time iterations).

Size	k_0	k_1	k_2	k_3	k_4
31	101.144	0.000	503.707	0.000	0.000
61	195.574	0.000	1478.746	0.000	0.000
91	314.110	0.798	4268.800	0.000	0.000

Table 5: Coefficients for the model $E_{time}(P) = k_0 + k_1 * P + k_2 * 1/P + k_3 * \log P + k_4 * P * \log P$ on the Network of Sun Hyper-Sparcs.

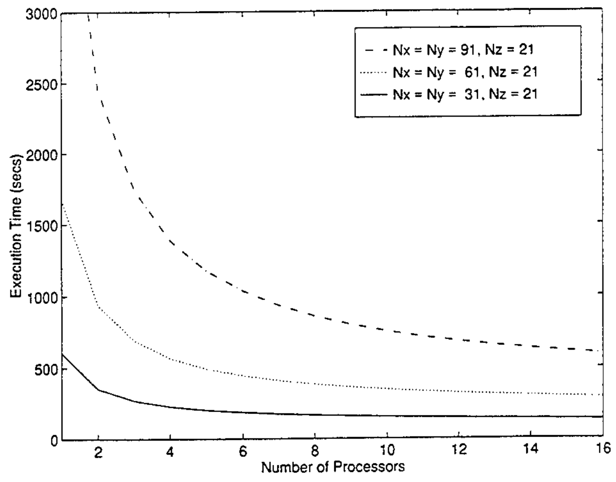


Figure 9: Processor Models on the Network of Sun Hyper-Sparcs (1000 time iterations).

time in terms of the number of processors P and the problem size N_x is:

$$E_{time} = 5.538 - 0.227 * N_x + 1.366 * P$$
$$-1.639 * (N_x/P) + 0.119 * (N_x^2/P)$$

The average error in prediction for this model was about 10 percent. Figure 10 shows model for the program as both a function of problem size and number of processors.

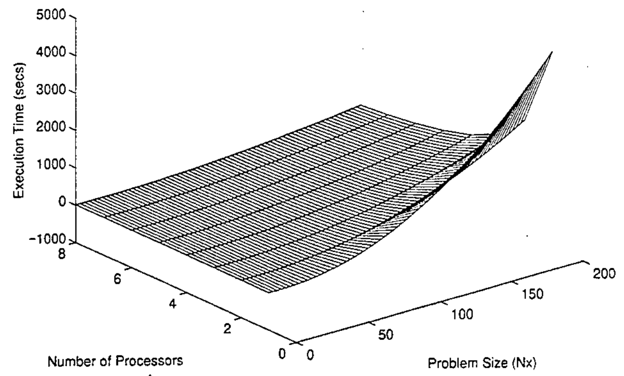


Figure 10: The Problem Size and Processor Model on the SGI Power Challenge, where $N_y = N_x$, $N_z = 21$ (1000 time iterations).

6 SUMMARY

An approach to modeling the performance of parallel programs was presented and applied to modeling the FDTD algorithm. The modeling technique is based on analyzing the structure of an existing program and measuring the performance of the program during actual runs on the target architecture as key factors of the application and architecture are varied. This information is used to create a closed-form expression for the execution time of the program in terms of those factors. The expression can then be used to predict the performance of the application over a wide range of problem size and number of processors.

The models showed that the FDTD application performed well when there was a sufficiently large problem size resulting in sufficient parallelism. The models showed that the execution time varied as a square of the problem size. This is due to the fact that the majority of the work was done within a triply nested loop, with the outer loop dimension (N_z) held constant. The models also showed a degradation in the performance in one of the configurations as the problem size increased. The reason for this was found to be non-unit stride accesses to memory. This was corrected by interchanging the rows and columns. This change resulted in a 30 percent performance improvement for $N_x = 131$.

7 REFERENCES

[1] V. D. Agrawal and S. T. Chakradhar. Performance estimation in a massively parallel system. In *Proceedings of Supercomputing' 90*, pages 306–313, 1990.

[2] J. P. Berenger. A perfectly matched layer for the absorption of electromagnetic waves. *Journal of Computational Physics*, 114:185–200, October 1994.

[3] R. J. Block, P. Mehra, and S. Sarrukai. Automated performance prediction of message-passing parallel programs. In *Proceedings of Supercomputing' 95*, 1995.

[4] E. A. Brewer, C. N. Dellarocas, A. Colbrook, and W. E. Weihl. Proteus: A high-performance parallel-architecture simulator. Technical Report MIT/LCS/TR-516, Massachusetts Institute of Technology, September 1991.

[5] R. Calalo, T. Cwik, N. Jacobi W. Imbriale, P. Liewer, T. Lockhart, G. Lyzenga, and J. Patterson. Hypercube parallel archtecture applied to electromagnetic scattering analysis. *IEEE Transactions on Magnetics*, 25:2898–2900, July 1989.

[6] W. C. Chew and W. H. Weedon. A 3d perfectly mached medium from modified maxwell's equations with strecthed coordinates. *IEEE Microwave and Guided Wave Letters*, 7:599–604, September 1994.

[7] M. J. Clement, M. R. Steed, and P. E. Crandall. Network performance modeling for pvm clusters. In *Proceedings of Supercomputing' 96*, 1996.

[8] R. Covington, S. Dwarakadas, J. Jump, S. Madala, and J. Sinclair. Efficient simulation of parallel computer systems. *International Journal on Computer Simulation*, 1(1):31–58, June 1991.

[9] M. E. Crovella and T. J. Leblanc. Parallel performance prediction using the lost cycles analysis. In *Proceedings of Supercomputing' 94*, pages 600–610, 1994.

[10] D. Culler, R. Karp, and D. Patterson. Logp: Towards a realistic model of parallel computation. In *Proceedings of the 4th ACM SIGPLAN Conference on Parallel Programming Practice and Experience*, pages 1–12, 1993.

[11] D. B. Davidson and R. W. Ziolkowski. A connection machine(cm-2). implementation of a three-dimensional parallel finite-difference time-domain code for electromagnetic field simulation. *International Journal of Numerical Modelling*, 8:221–232, August 1995.

[12] P. M. Dickens, P. Heidelberger, and D. M. Nicol. Parallelized direct execution simulation of message-passing parallel programs. *IEEE Transactions on Parallel and Distributed Systems*, 7(10):1090–1105, October 1996.

[13] M. A. Driscoll and W. R. Daasch. Accurate predictions of parallel program execution time. *Journal of Parallel and Distributed Computing*, 25:16–30, 1995.

[14] R. Kroger F. Lange and M. Gergeleit. Jewel: Design and implementation of a distributed measurement system. *IEEE Transactions on Parallel and Distributed Systems 3*, pages 657–671, November 1992.

[15] R. D. Ferraro. Solving partial differential equations for electromagnetic scattering problems on coarse-grained concurrent computers. In T. Cwik and J. Patterson, editors, *Computational Electromagnetics and Supercomputer Architecture*, number 7, pages 111–154. EMW Publishing, Cambridge, MA, July 1993.

[16] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard. Technical Report Computer Science Department CS-94-230, University of Tennessee, Knoxville, TN, 1994.

[17] D. Gannon, F. Bodin, S. Srinivas, N. Sunderasan, and S. Narayana. Sage++, an object oriented toolkit for program transformations. Technical report, Department of Computer Science, Indiana University, 1993.

[18] S. D. Gedney. Finite-difference time-domain analysis of microwave circuit devices on high performance vector/parallel computers. *IEEE Transactions on Microwave Theory and Techniques*, 43:2510–2514, October 1995.

[19] S. D. Gedney. An anisotropic perfectly matched layer absorbing media for the truncation of fdtd lattices. *IEEE Transactions on Antennas and Propogation*, 44:1630–1639, December 1996.

[20] S. D. Gedney. An anisotropic pml absorbing media for fdtd simulation of fields in lossy dispersive media. *Electromagnetics*, 16:399–415, July/August 1996.

[21] S. D. Gedney. Efficient implementation of the uniaxial pml absorbing media for the finite-difference time-domain method. *13th Annual Review of Progress in Applied Computational Electromagnetics*, 2:892–899, 1997.

[22] S. D. Gedney. The perfectly matched layer absorbing medium. In A. Taflove, editor, *Advances in Computational Electrodynamics: The Finite-Difference Time Domain.* Artech House, Boston, 1998.

[23] S. D. Gedney and S. Barnard. Efficient FD-TD algorithms for vector and multiprocessor computers. In A. Taflove, editor, *Finite Difference Time Domain Methods for Electrodynamic Analysis*, Boston, MA, 1995. Artech House.

[24] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *Parallel Virtual Machine: A User's Guide and Tutorial for Networked Parallel Computing.* The MIT Press, London, England, 1994.

[25] A. Geist, J. A. Kohl, and P. Papadopoulus. Visualization, debugging and performance in pvm. In *Proceedings of Visualization and Debugging Workshop*, October 1994.

[26] J. K. Hollingsworth and B. P. Miller. The Paradyn Parallel Performance Tools and PVM. *Enviroments and Tools for Parallel Scientific Computing*, 1994.

[27] Raj Jain. *The Art of Computer Systems Performance Analysis.* John Wiley & Sons, Inc., 1991.

[28] D. S. Katz, E. T. Thiele, and A. Taflove. Validation and extension to the three-dimensions of the berenger pml absorbing boundary condition for fd-td meshes. *IEEE Microwave and Guided Wave Letters*, 4:268–270, August 1994.

[29] W. Meira, Jr. Modeling performance of parallel programs. Technical Report 589, The University of Rochester, Computer Science Department, Rochester, New York 14627, June 1995.

[30] J. Patterson, T. Cwik, R. Ferraro, N. Jacobi, P. Liewer, T. Lockhart, G. Lyzenga, J. Parker, and D. Simoni. Parallel computation applied to electromagnetic scattering and radiation analysis. *Electromagnetics*, 10:21–40, January - June 1990.

[31] S. Reinhardt, M. Hill, J. Larus, A. Lebeck, J. Lewis, and D. Wood. The Wisconsin Wind Tunnel: Virtual prototyping of parallel computers. In *Proceedings of the ACM SIGMETRICS Conference*, pages 48–60, May 1993.

[32] Z. S. Sacks, D. M. Kingsland, R. Lee, and J. F. Lee. A perfectly matched anisotropic absorber for use as an absorbing boundary condition. *IEEE Transactions on Antennas and Propogation*, 43:1460–1463, December 1995.

[33] S. R. Sarukkai. Scalability analysis tools for SPMD message-passing parallel programs. In *Proceedings of Mascots 94*, pages 180–186, January-February 1994.

[34] C. H. Sauer and K. M. Chandy. *Computer Systems Performance Modeling.* Prentice Hall, 1981.

[35] J. Schneider. FDTD BibTex Web Page. In *http://www.eecs.wsu.edu/˜schneidj/fdtd-bib.html*, 1996.

[36] A. Sivasumbramaniam, A. Singla, U. Ramachandran, and H. Venkateswaran. A simulation-based scalability study of parallel systems. *Journal of Parallel and Distributed Computing*, 22(3):411–426, September 1994.

[37] A. Taflove. *Computational Electrodynamics: The Finite-Difference Time-Domain Method.* Artech House, Boston, MA, 1995.

[38] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

[39] J. Yan. Performance Tuning with AIMS–An Automated Instrumentation and Monitoring System for Multicomputers. In *proceedings of the 27th HICS, Wailea, Hawaii*, January 1994.

[40] K. S. Yee. Numerical solution of initial boundary value problems in isotropic media. *IEEE Transactions on Antennas and Propogation*, AP-14:302–307, May 1996.

Implementation and Application of a FD-TD Simulation Tool for the Analysis of Complex 3D Structures

Kevin Thomas
Cray Research
655 Lone Oak Road, Eagan, MN 55121
phone 612-683-3624, fax 612-683-3099
kjt@cray.com

Gary Haussmann and Melinda Piket-May
Department of Electrical and Computer Engineering
Campus Box 425, University of Colorado, Boulder,CO 80309
phone 303-492-8719, fax 303-492-5323
Gary.Haussmann@Colorado.edu
mjp@boulder.colorado.edu

Roger J. Gravrok
Sequent Computer Systems
Bldg. D2, Ste. 219 Mailbox 82
800 Wisconsin Street
Eau Claire, WI 54703-3611
phone 715-830-7321, fax 715-833-2027
rjg@sequent.com

Abstract

This paper presents information about the development of an electromagnetic analysis tool "LC" which integrates the Finite-Difference Time-Domain (FD-TD) method with an interactive Graphical User Interface (GUI). The paper will discuss the program implementation and design; many issues in the implementation have surfaced, concerning the problem of producing a graphical model editor/simulator that runs efficiently on various hardware systems. The paper will also explore how the solver's capabilities aid design engineers when investigating and solving packaging and interconnect design issues as well as the program's application to engineering problems.

1 Introduction

At the level of a chip, or a device comprised of several chips with interconnects, the electromagnetic environment is very complex and not necessarily well understood theoretically, numerically, or experimentally. For instance, coupling between vias can distort signals, mismatches between vias and signal lines can lead to ground-bounce, holes in ground planes may result in increased coupling effects between board layers, and metal traces are likely to have reactive impedance components that can degrade system performance at higher clock speeds. The simulation tool discussed in this paper was designed to aid in understanding these effects, as well as to discover means to mitigate and/or overcome these effects. Also, this paper will discuss the development of general methods used to extract relevant information from the simulation.

2 Background

We have chosen to use the Finite-Difference Time-Domain (FD-TD) method as our Maxwell's Equations solver [1]. The FD-TD method is a very powerful computational tool: it gives time-domain data, useful for transient analysis, and can yield frequency-domain data via Fourier Transforms. Taken together, all of these capabilities have made FD-TD an attractive and robust method for solving a number of electromagnetic interaction problems. Previously a major limitation to the full scale industrial deployment of FD-TD tools has been the absence of a versatile GUI to generate models and postprocess the vast amount of data that the FD-TD method can generate.

This paper will discuss an EM analysis development tool called "LC". It consists of a GUI interface with the ability to auto-mesh a graphically defined three-dimensional user geometry. A number of different excitations are possible, and the outputs can yield not only field data, but also voltages, currents, impedances, inductances, capacitances and fluxes. Further, both time and frequency-domain data are available on output. LC also allows for 2-D visual plots of the full 3-D problem being simulated during time-stepping. The ability to visualize the three-dimensional electrodynamics of a structure lends a great deal of intuitive insight and understanding to the user. This is also very useful in identifying problem areas with a particular design.

Another very important and relevant feature of the LC tool is an interface to SPICE [2], which will analyze a circuit based on FD-TD field values linked to SPICE at each time step. The SPICE interface also generates output voltages and/or currents that modify the FD-TD field values which are then used during the next FD-TD time-step. The efficient implementation of LC on scalable parallel computers (under development) will allow for the examination of problems that are currently too computationally large or complex, conceivably making inroads into some of the grand challenges facing the electromagnetic engineering community.

Essentially LC is an integrated EM model editor, simulator, and analysis tool. It's composed of 150,000 lines of C and Fortran and is written to run on Unix. The simulator can use multiple processors in parallel to reduce the solution time on shared-memory multiprocessors. A full description can be found on the LC web page *http://www.cray.com/lc/*. While the program source code is not public domain, a demonstration version (distributed in executable form) is available for many types of computers.

The LC tool is so named because it's initial development was driven by the need for an accurate three-dimensional characterization of the inductance (L) and capacitance (C) of complex electronic systems. Specifically, the creation of LC was motivated by the need to develop techniques and a corresponding tool that were both accurate and suitable for the analysis/design of circuits with clock speeds above 100 MHz. Beyond 100 MHz, circuit interconnect performance dictates system performance capabilities. Accurate electrical characterization of complex electronic structures in three dimensions is mandatory for good design. EM modeling challenges such as those associated with printed circuit boards, multichip modules, integrated circuit packages, connectors, and electromagnetic interference are currently being performed. LC may be used to look at signal- transmission characterization and power distribution modeling. The characterization requirements for these problems include an intuitive and visual understanding of the electrodynamics along with a quantitative description of L,C,R,Z,and T_d parameters.

3 Implementation Details

This section describes the FD-TD details involving mesh setup, boundary conditions, parallelization, and optimization/performance testing. The bulk of the FD-TD engine is based on standard FD-TD updates [1], with slight modifications to allow for straightforward programming to communicate to the user interface code. Other key modifications include the ability to choose from numerous boundary conditions and parallelization of the original serial code.

3.1 Meshing

The FD-TD algorithm in LC is based on a three dimensional Cartesian grid. A rectangular zone enclosing the model space is discretized into uniformly-sized cubic cells; thus a mapping is achieved to an *ijk* (integer-indexed) 3-D grid. Each 3-D cell is assigned a single material definition, based on the material at the center of the cell, making it homogeneous. Each cell is broken down into its six faces, and a material precedence and averaging scheme is applied to the faces of adjoining cells. In the final meshing step, the faces are broken down into edges. During this meshing procedure, the properties of the materials throughout the grid are converted into electric and magnetic field coefficients assigned to the edges and faces of the cells, respectively.

3.2 Boundary Conditions

Special boundary conditions are usually applied to the fields on the faces of the computational grid. The default boundary condition in LC is the first order Mur absorbing boundary condition [3]. This boundary condition absorbs outgoing energy, in effect extending the edge of the grid indefinitely. The first order Mur introduces very little computational overhead for this benefit, but is only suitable for a restricted set of problems.

The Berenger perfectly matched layers (PML) boundary condition [4, 5] is also available in LC. It is a much more general absorbing boundary condition, and is useful for models with complex materials and geometries. Although it introduces a large amount of computation to the simulation, the algorithm has two useful features for managing the load. First, it is structured in a very analogous way to the FD-TD algorithm. Thus, most of the same techniques for efficiently updating cells in the main FD-TD also apply to the PML region. Second, the number of layers in the absorbing region is user-defined. The number of layers directly influences the amount of memory and computation required for the boundary condition. It also determines the accuracy (its ability to absorb outgoing energy).

3.3 Parallel Code Development

The objective of this effort is to fully develop an interactive electromagnetic (EM) design and analysis tool (LC) that will take complete advantage of the coming class of scalable parallel computers. The efficient implementation of this tool on scalable parallel computers will allow for the examination of problems that are currently too computationally large or complex, conceivably making inroads into some of the grand challenges facing the electromagnetic engineering community.

As part of a contract with NASA through the EMCC (Electromagnetics Code Consortium), Cray developed a parallel FD-TD application program for computing RCS (radar cross section). Implemented using PVM (Parallel Virtual Machine) message passing, the resulting software is portable to all HPC hardware. Code validation and testing proved that the software achieved over 50% of the theoretical peak performance, while scaling linearly up to hundreds of processors [6].

We have leveraged this experience in developing our scalable parallel implementation of LC. The standard version of LC uses implicit parallelism (via compiler directives) to achieve parallel speedup on the Cray shared-memory vector machines, and on smale-scale paralle workstations, such as the Octane and Origin 200. An explicitly parallel implementation of LC, implemented using the parallel Message Passing Interface (MPI) [7], is in progress. This explicitly parallel version currently does not allow for the full interactive GUI present in the standard version of LC, but will correctly process and simulate all standard structures and materials present in the GUI version of LC.

The parallel bookkeeping and passing of data between processors is done using the parallel Message Passing Interface (MPI) [7]. MPI was chosen because it provides good performance combined with portability, allowing the parallel version of LC to run on a wide variety of architectures just as the original serial version of LC did. MPI also has built in functionality to configure the parallel machine as a Cartesian array (i.e., as an $N \times M \times P$ array of processors, where N,M,and P are user selected), in whatever manner is appropriate for the current hardware. This is achieved using a "virtual Cartesian array," created by the `MPI_CREATE_CART` function.

Much of the information about the simulation, such as the grid size and shape, is not known explicitly but deduced at run time. The user can specify the grid size and shape, as well as the structure to analyze at run-time, just as the program is starting execution. If instead the simulation parameters (material values needed, waveform frequency and shape, etc.) were found at compile time and not run time, then every time a parameter changed the user would need to enter all these parameters, recompile the program, and finally run the simulation. By delaying these choices until run time the user is spared a good deal of time and overhead normally involved in the programming/compiling phase.

However, the same run time properties that make design easy for the user make implementation hard for the programmer. If everything had been specified at compile time, such issues as dividing up the problem amongst multiple processors, and inter-processor communication, would have been examined and solved at compile time. However, because the program does not have enough information at compile time, it must wait until run time. As it is running the program must properly redistribute the problem over all of its processors, set up communication channels to transfer data between processors, and funnel the analysis results into various (one to many) output data files. It is this process of data decomposition and inter-process communication that presents the most implementation problems.

The internal data layout of LC records the simulation structure as a list of "blocks" which can represent a material, data probe, source excitation, or several other objects. The overall structure being studied is manipulated by adding, examining, and destroying these blocks. Just before a simulation is run, the representation is frozen and converted into an FDTD mesh, along with some extra data representing the sources, probes, and other information not directly stored in the FDTD mesh. This conversion process—the transforming of a block list into an FDTD mesh, called "meshing" in the program—is where most of the actual programming problems arise for the parallel implementation.

Once the problem is partitioned onto the parallel processors, meshing can proceed just as it did in the serial version. However, one cannot use the original block list on each processor, because then each processor would be meshing and simulating the entire grid. Therefore, each processor obtains a private copy of the block list and modifies that copy to contain only relevant blocks (figure 1). Each processor is responsible for computing a certain region of the total grid; any blocks outside that region are discarded and the remaining blocks are clipped so that they are totally contained in the processor's computational region.

After the block list on each processor has been modified, it is processed normally by using the original serial meshing algorithm. Since the block list only specifies blocks within the processor's computational region, the meshing algorithm will only allocate and mesh that region. The process occurs on all the processors in parallel, with each transforming the block list into a local grid. Proper manipulation of the block list before this step insures that no two processors try to allocate and simulate the same region,

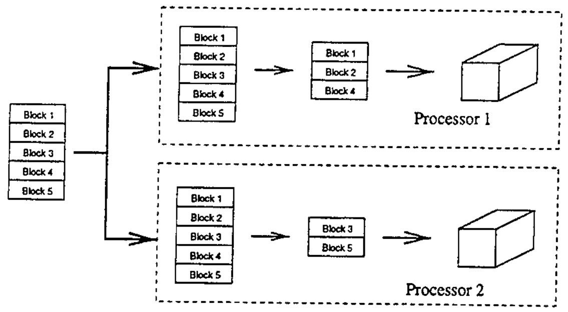


Figure 1: Each processor obtains and clips its own copy of the block list

or overlapping regions.

The process of discarding and clipping elements in the block list will automatically divide up material blocks correctly, so the overall structure under study will be properly partitioned and meshed on the parallel array of processors. However, non-material elements will not function correctly once they have been split onto multiple processors. The serial code will process a given source excitation or data probe block as if it existed only on that processor; if a data probe block is split across two processors, for instance, then each processor will collect probe data for its "local" probe. Some mechanism must exist to gather up the data from all the local probes into the larger probes in the original block list.

Similarly, the source excitation blocks must be configured to spread their voltage or current amplitude across multiple processors. These source blocks are modified to generate a smaller current or voltage, based upon their new size; for instance, if voltage excitation is split exactly in half across two processors, then each of these two new blocks will now generate half the original voltage. Voltage excitations split in other ways will generate voltage amplitudes in proportion to their new size.

Data probe blocks gather their data locally on each processor, and must somehow combine the data into a final, single value. The parallel implementation does this by collecting probe information data from every processor. This probe information lists what probes exist on what processors. If an element such as a data probe spans multiple processors, the element is divided into pieces: one piece of the probe goes onto each processor. During the simulation the pieces combine their data so that they appear and generate output as if they were one single large probe (figure 2. This data is then written out to disk storage or visualized for the user, as it was in the original serial LC implementation.

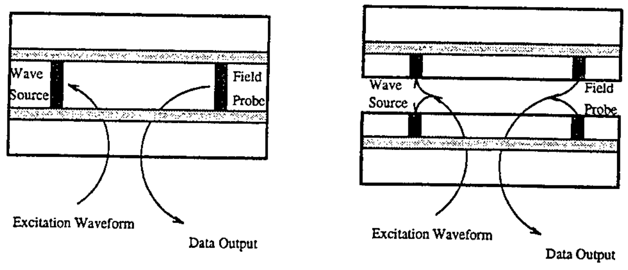


Figure 2: Splitting of arbitrary elements (wave sources and probes) across processor boundaries. Compare a simulation running on one processor (left) and distributed on two processors (right). Arrows show data flow between the processors running the simulation and external stimulus/data storage

3.4 Optimization

Many techniques to speed the FD-TD time-stepping process have been tried in LC. During each half time-step, every field update is independent of the others. Thus substantial speedups can be achived by overlapping the field updates. Vector computers are effective when the grid dimensions are large, because the field updates form a long vector update, hiding memory access time. If the fields are updated in a pattern which results in consecutive memory addresses being used, optimum memory access patterns can be achieved. In Fortran, this means that the inner-most loop of the grid update varies the left-most array index. Cache-based computers with long cache lines, like the IBM RS/6000, this pattern effectively uses the computer's ability to overlap memory access with computation.

Efficient use of a computer requires that the computational requirements are balanced with the memory bandwidth requirements. The large memory bandwidth requirement of FD-TD limits its performance. In a simple FD-TD field update, seven operands are fetched, and one result stored, with only six operations performed. Many computers have the ability to combine a multiply-add operation of the form $a * b + c$ into a single operation. This ability reduces the effective number of operations to four; thus, two values must move for every operation performed.

The number of operands which are read from memory can be reduced while updating a homogeneous region of the grid. This optimization decreases the memory bandwidth requirement relative to the number of operations. In this case, two operand fetches from main memory can be eliminated, reducing the total to five per update. LC automatically detects

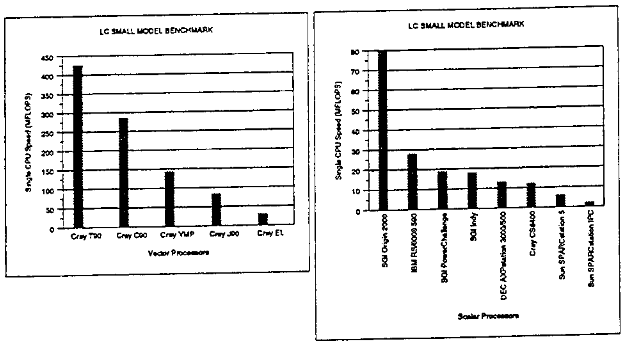


Figure 3: LC electromagnetic simulation performance

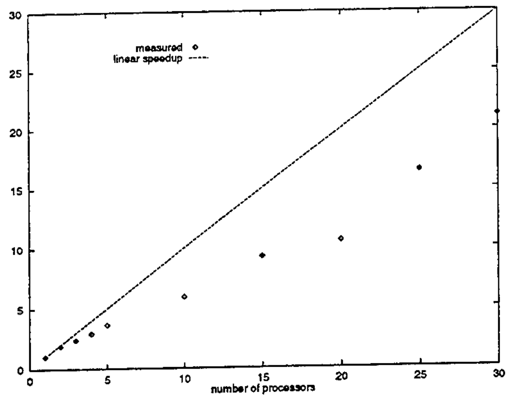


Figure 4: Comparison of ideal (linear) speedup to measure speedup. Measurements were made on a 127 processor Cray Origin 2000 computer.

global conditions of this type and employs an optimized update.

Since the field updates are independent, parallel FD-TD is straightforward. The grid can be divided into as many regions as necessary, with each region assigned to a processor. In a shared memory computer, the only limiting factor is the size of the regions. If the regions become too small, the processors have insufficient work available between synchronization points. In a distributed memory computer, an additional constraint of communication overhead is present; the field values in the adjacent region are required when updating fields on the edge of a region. Again, when the regions are large, this communication is a small percentage of the overall run time. A hybrid computer architecture, such as the Cray Origin 2000, can be programmed efficiently as a shared memory computer as long as the grid regions are much larger than the hardware page size; most of the memory will reside on the local compute node, with inter-node communication occurring at the region and page boundaries.

3.5 Performance

To obtain some insight into the performance characteristics of the FD-TD solver in LC, a benchmark problem was created. It was designed to be small enough to be practical to run on a desktop computer. This benchmark structure is a microstrip running over a rectangular hole in a groundplane; the size of the simulation is approximately 1.3 Megabytes on architectures using a 32-bit floating point representation. The benchmark was run on as many computers as available, and the results are shown in figure 3. For the multi-CPU computers, the speeds shown are for a single CPU.

Early tests using the MPI explicitly parallel form of LC on a small (8 processor) Origin 2000 show a linear to superlinear speedup on a small microstrip simulation. Larger applications do not achieve superlinear performance, but show better scalability than the smaller applications (figure 4). The break point at 20–25 processors is the point where each processor can fit all of its data in local cache, explaining the sudden slope increase in the curve at 25+ processors.

4 Modeling and Parameter Extraction of 3-D Structures

In the current work, a priority was given to three-dimensional modeling accuracy. The current techniques employ a simple method that uses the characteristic definitions of inductance to quantitatively determine the true inductive response directly from the field data. Full-wave solutions to Maxwell's equations are used within the FD-TD engine to accurately determine the true return-path distributions. The user controls the tradeoff between accuracy and computational-resource usage.

4.1 Modeling Signal Path Inductance

Direct calculation of the equivalent-circuit inductance (self or mutual) of any three-dimensional path is obtained with a simple application of first-principles of electromagnetics [8].

Inductance is defined as the ratio of the magnetic flux ϕ through a surface S that links a current I flowing through the surface. The current flowing through the surface S is found by integrating the magnetic field $\vec{H}$ on the closed contour around the surface.

$$\mathcal{L} = \frac{\phi}{I} = \frac{\int_S \vec{B} \cdot ds}{\oint_C \vec{H} \cdot dl} = \frac{\int_S \mu \vec{H} \cdot ds}{\oint_C \vec{H} \cdot dl} \qquad (1)$$

In the FD-TD grid, these integrals translate into discrete sums of field values in the FD-TD grid, approximating the continuous field values given in equation (1).

$$\mathcal{L} = \frac{\mu \sum \sum_S \vec{H} \cdot \Delta^2}{\sum_C \vec{H} \cdot \Delta} \qquad (2)$$

Where $\Delta = \Delta x = \Delta y = \Delta z$. If the measurements of magnetic flux ϕ and current I are made on a finite length l of transmission line, the inductance per unit length $\mathcal{L}'$ is found from the inductance $\mathcal{L}$ by

$$\mathcal{L}' = \frac{\mathcal{L}}{l} \qquad (3)$$

The capacitance is defined as the ratio of the charge in a volume enclosed by a surface to the electromagnetic potential of the surface. The charge in a volume is found by integrating the electric field on the surface enclosing that region, and the potential is found using a line integral of the electric field:

$$\mathcal{C} = \frac{Q}{V} = \frac{\oint_S \vec{D} \cdot ds}{\int_C \vec{E} \cdot dl} = \frac{\oint_S \epsilon \vec{E} \cdot ds}{\int_C \vec{E} \cdot dl} \qquad (4)$$

In the FD-TD simulation, these integrals translate into discrete sums of fields values taken from the FD-TD grid:

$$\mathcal{C} = \frac{\sum \sum_S \epsilon \vec{E} \cdot \Delta \cdot \Delta}{\sum_C \vec{E} \cdot \Delta} \qquad (5)$$

For example, given a two conductor transmission line as shown in Figure 5, the flux integral would be performed on the surface S with the direction of the surface vector dS as shown. The current integral would be performed along the contour C.

With this method, the challenge of modeling three-dimensional inductance simply reduces to the user properly defining the closed surface over which the flux integration is performed. Call this closed surface the flux net S. The incremental inductance for the segment of trace 1 from A to B is calculated using the flux net defined by the plane ABCD. The return current flows from C to D. The upper and lower edges of this flux net are defined by the edges of the conductors that carry the signal and return-path currents.

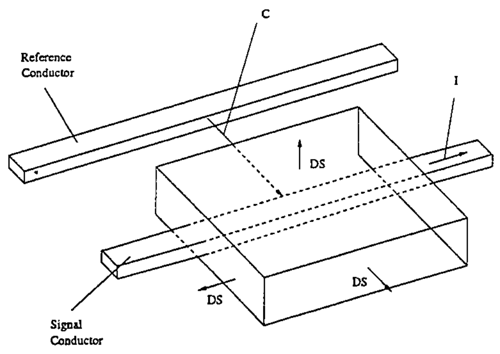


Figure 5: Inductance calculated from flux through S and current on the transmission line

Mutual inductance between two traces, 1 and 2, is modeled by calculating the total amount of flux caught in the flux net defined by ABCD when the signal current I_2 is sent down trace 2. The above illustrative example was simple and two-dimensional in nature. Consider how this simple method is used to quantify the dramatic increase in effective trace inductance when a slot is cut in the ground plane. Now the return path current is diverted around the slot. This new return path converts the simple planer flux net in figure 5 to a hockey net configuration. An example simulation of the above structures uncovers a tripling of effective trace inductance due to the slot in the ground plane.

4.2 Inductance Modeling of three-dimensional Power-Distribution Structures

Determining the equivalent lumped inductance of a three-dimensional structure is of critical importance to electrical engineers involved in package design. Physical and electrical design restrictions can be met and optimal performance can be achieved if this inductance is known before the prototyping stage. Signal-path inductance can be modeled with $L = \frac{\phi}{I}$. In this approach, knowledge of the ground current I return path is required to extract the flux ϕ. This is a fairly straightforward extraction for simple structures, such as a coaxial cable. For more complex models this return path current, and therefore the inductance, is not as easily determined. One such example of a structure without a simple equation for inductance is a power distribution system composed of numerous meshed ground/power planes with connecting vias. This inductance method utilizes a system-level approach involving the analysis of the voltage and current waveforms associated with the structure's inductive element.

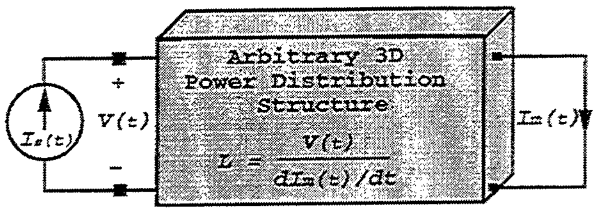

Arbitrary 3D Power Distribution Structure
$L = \frac{V(t)}{dI_2(t)/dt}$
+
$V(t)$
−
$I_1(t)$
$I_2(t)$

Figure 6: Power distribution structure.

This novel "black-box" method of solving for distributed structure inductance is visualized in the block diagram of (figure 6). In this configuration, an input signal is excited and the voltage (V) across the input is measured. The signal path is shorted out at the output, where the current (I) is measured. With these two sets of data available, the inductance of the system can now be calculated using equation (6):

$$L = \frac{V}{\frac{\delta I}{\delta t}} \tag{6}$$

This approach does not depend on the interior composition of the structure itself, but rather on the ratio of the voltage across the input to the output current time derivative. The voltage-to-current relationship (shown in figure 7) develops as a direct result of the structure's inductive element.

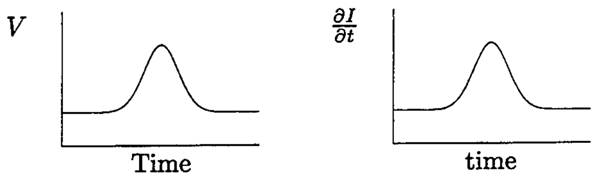


Figure 7: Inductance voltage-to-current relationship; Gaussian pulse input signal.

4.2.1 Meshed PCB Model

A significant problem in modeling and designing high speed digital applications has been understanding how to extract system-level parameters from complex distributed systems. One such complicated structure is shown in figure 8. This structure is a meshed pcb system of three power planes with current sources at the edges, five ground planes, and several vias connecting the power and ground planes. The model

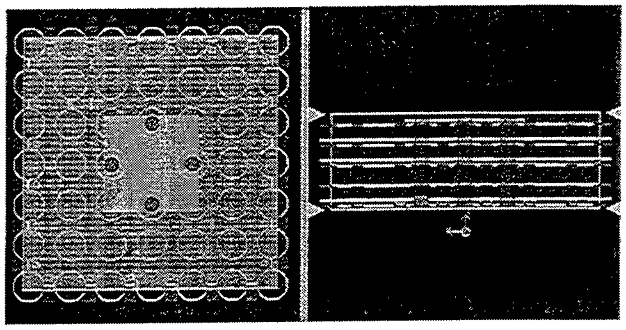


Figure 8: Meshed pcb structure modeled in FD-TD interface

is based on an actual power distribution structure. When the planes are meshed, approximately 50% of the metal is removed. The input voltage waveform is measured between a power plane and a ground plane. A via short to a ground plane on top of the structure provides the current response which is due to the structure's inductive element. Until now, the inductance calculation for this type of structure has been done on a piecewise scale; that is, by examining the effects of the power/ground planes on the vias and vice versa.

However, by applying this new system-level method to the system, it is possible to represent the entire structure as a lumped inductor with one straightforward calculation. After running the FD-TD simulation and extracting the required voltage and current measurements, the inductance of a 3mm × 3mm subsection of a PCB was found in the numerical model to be 83.8pH.

It is clear that this approach can be useful in determining the optimal physical parameters of a system while staying within electrically-dictated inductance thresholds that often accompany high speed digital applications. This method can be integral in the development of certain design guidelines for structures similar to the one in figure 8.

5 Conclusion

This work has direct application for analysis and design of advanced electronic components such as multichip modules (MCM) and mixed signal modules. The capabilities of LC address requirements that are currently being pursued in both the defense and commercial sectors. It is directly applicable to existing efforts in design and analysis of advanced packaged modules, low cost electronic packaging, phased array antennas, MCMs, mixed signal modules, and the study of electromigration phenomenology. We eventually envision a tool that will not only be able to address the complex electromagnetic interactions within individual devices and components but will also address coupling of energy into electronic devices from external sources (for example noise or jamming signals).

We are planning to investigate the use of grid partitioning to employ specialized field updates to reduce the amount of memory and memory bandwidth required during the simulation. Using this technique, the amount of memory required can be reduced by up to two thirds, and the processing speed can be increased by a factor of two..

We plan to add new algorithm and postprocessing features that are requested by users. One algorithm directly relevant to modeling power and signal distribution structures involves taking into account losses due to skin effects. Another one is the modeling of dispersive materials [9, 10, 11].

LC has been the culmination of approximately a decade of electromagnetics research and development at Cray. This experience includes CRADAs with National Labs, government funding of Radar Cross Section (RCS) work, and code development at Northwestern University and the University of Colorado at Boulder.

References

[1] Allen Taflove, *Computational Electrodynamics the Finite-Difference Time-Domain Method*, pp. 98–100,111–134,281–295, Artech House, Boston, 1995.

[2] V.A. Thomas, M.E. Jones, M.J. Piket-May, A. Taflove, and E. Harrigan, "The use of spice lumped circuits as sub-grid models for fd-td analysis," *IEEE Microwave and Guided Wave Letters*, vol. 4, pp. 141–143, 1994.

[3] G. Mur, "Absorbing boundary conditions for the finite-difference approximation of the time-domain electromagnetic field equations," *IEEE Trans. on Electromagnetic Compatibility*, vol. 23, pp. 377–382, 1981.

[4] D.S. Katz, E.T. Thiele, and A. Taflove, "Validation and extension to three dimensions of the berenger pml absorbing boundary condition for fd-td meshes," *IEEE Microwave and Guided Wave Letters*, vol. 4, pp. 268–270, 1994.

[5] J.P. Berenger, "A perfectly matched layer for the absorption of electromagnetic waves," *Journal of Computational Physics*, vol. 114, pp. 185–200, 1994.

[6] S. Barnard and A. Taflove, "Application of massively parallel supercomputing to 3D RCS methods and modeling of complex materials," Final report on nasa-ames contract nas213890, Dec. 1994.

[7] "The message passing interface (MPI) standard 1.2 and 2.0," http://www.mpi-forum.org/.

[8] Roger Gravrok, Melinda Piket-May, and Kevin Thomas, "LC: An integrated methodology to model and visualize the complex electrodynamics of 3D structures," in *Electrical Performance of Electronic Packaging*. IEEE MTT Society, Oct. 1995.

[9] X. Zhang, J. Fang, K.K. Mei, and Y. Liu, "Calculation of the dispersive characteristics of microstrips by the time-domain finite-difference method," *IEEE Trans. on Microwave Theory Tech.*, vol. 36, pp. 263–267, 1988.

[10] T. Kashiwa and I. Fukai, "A treatment by FDTD method of dispersive characteristics associated with electronic polarization," *Microwave and Optics Technologies Letters*, vol. 3, pp. 203–205, 1990.

[11] R. Luebbers, F. Hunsberger, K. Kunz, R. Standler, and M. Schneider, "A frequency-dependent finite-difference time-domain formulation for dispersive materials," *IEEE Trans. on Electromagn. Compat.*, vol. 32, pp. 222–229, 1990.

Optimizing the Parallel Implementation of a Finite Difference Time Domain Code on a Multi-user Network of Workstations

J.V. Mullan, C.J. Gillan and V.F. Fusco

The High Frequency Electronics Laboratory
Dept. Electrical and Electronic Engineering
The Queen's University of Belfast
Ashby Building, Stranmillis Road, Belfast
N. Ireland BT9 5AH, UK

Abstract

The implementation of a parallel, three dimensional, finite difference time domain (FDTD) computer program is considered and applied to a test scattering problem on a multi-user network of desktop workstations. The computation has primarily been done on a local area network (LAN) using six identical HP 9000/715 workstations (i.e. a homogeneous environment) with the Parallel Virtual Machine (PVM) software being employed as the communications harness.

In this paper the sequential and parallel FDTD approaches are reviewed. We investigate the factors which cause a reduction in efficiency in the latter, such as host allocation and load balancing. We propose a task migration process, which is efficient for the FDTD algorithm, as a partial solution. The advantages of this approach are discussed and further developments based on available computational resources are suggested.

Introduction

The finite difference time domain method (FDTD), as formulated by Yee[1], is now a well established, numerical and storage intensive, approach to solving Maxwell's equations for the electromagnetic field. The FDTD method has been employed, for example, in the design of microwave circuits[2], in radar cross section prediction[3] and in antenna design[4]. Presently, the method is being used extensively in investigations of biological interaction with electromagnetic fields[5]. The modelling of the fields radiated by a mobile telephone, when it is close to the human head, is a very active area of research at this time. Not least due to the complexity of human anatomy and physiology, the latter interaction necessitates that a parallel algorithm be used in order to solve the FDTD equations within a reasonable amount of time with available resources.

The *non-sequential* FDTD method has been investigated on a variety of hardware including transputers[6, 7], vector computers[8], massively parallel processors[8, 9] and networks of workstations[10, 11]. Excluding vector computation, the general strategy has been to apply a straightforward geometric decomposition of the total volume, an approach that is also employed in other areas of technical computing, for example molecular dynamics[12]. In order to achieve optimum geometric decomposition, the topology of the connections between the processors working concurrently on the problem must map the geometry of the data-domain that the problem is defined upon.

In this paper we follow the approach of Chew and Fusco, this time operating on a network of HP9000/715 workstations with the aid of the Parallel Virtual Machine (PVM) software[13]. The test problem simulates

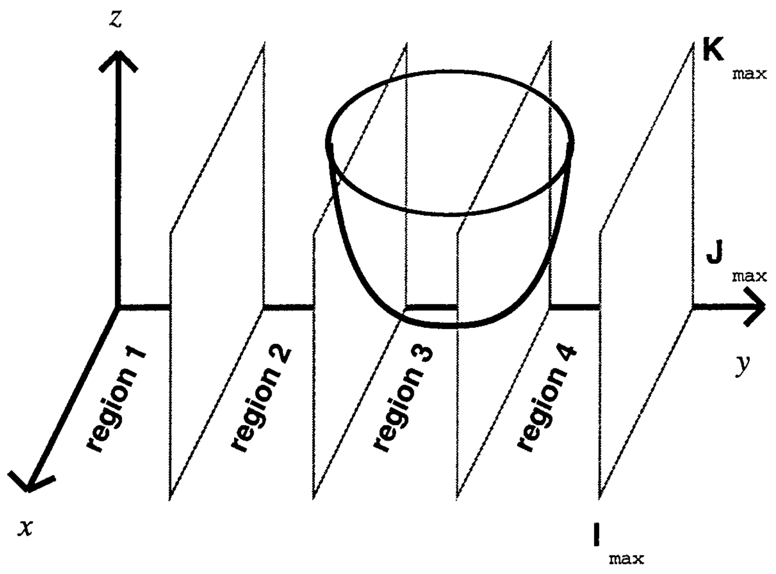


Figure 1: Segmentation of solution region into N subspaces by partitioning the grid along one dimension. In this case, the y-axis is partitioned and $N = 4$.

the scattering of a 2.5 GHz sinusoidal plane wave by a lossless dielectric sphere ($\epsilon_r = 4.0$), where the rest of the computation space consists of air. Timestepping continues until steady state conditions are achieved in each Yee cell; owing to symmetry only one quarter of the problem needs to be evaluated. This simple test case was used so that the efficiency of this method could be assessed and optimized before being used on more complex and useful calculations.

The sequential and parallel FDTD Algorithms

Other papers in this volume present the FDTD algorithm in considerable detail. However, for the sake of completeness, we give the salient details for a sequential implementation of the method here in order to facilitate an explanation of the way in which this is enhanced to achieve a parallel implementation.

The finite difference approach to solving Maxwell's equations for the electromagnetic field, as a function of position and time, approximates the infinite space-time continuum by a discrete three dimensional spatial grid of finite extent on which the electromagnetic field components are updated at successive, discrete time steps, that is on a temporal grid. The partial derivatives occurring in Maxwell's equations are approximated by difference equations defined with respect to the spatial and temporal grid. Yee's[1] approach is to treat Maxwell's curl equations as a pair of coupled, first order, partial differential equations and uses a specific griding arrangement, motivated by simple electromagnetic principles, which makes the difference approximations accurate to second order. Although this it is by no means the only solution strategy, or choice of grid structure, the Yee method quickly became a de-facto standard in computational electromagnetics[8].

The Yee algorithm is expressed as a loop over a finite number of timesteps where the work done at each timestep is as follows:

1. Update the E-field components, a task which uses only previously computed E and H-field components. The update, at each point on the spatial grid, is therefore *independent* of updates at other points on the spatial grid, for this timestep, and requires only field values from adjacent Yee cells due to the

form of the finite difference equations.

2. Augment time by one half of a time step.

3. Update the H-field components, a task which uses the E-field components computed in 1 above as well as only the previously computed H-field components. Again, the update, at each point on the spatial grid, is *independent* of updates at other points on the spatial grid, for this timestep, and requires only field values from adjacent Yee cells due to the form of the finite difference equations.

4. Apply outer radiation boundary conditions to the surfaces of the finite volume in order to compensate for truncation of the infnite spatial continuum. The boundary condition is applied in a point by point fashion with the computations at each point being independent but requiring results from step 1 above.

Clearly, the Yee algorithm exhibits a high degree of concurrency, a feature that cannot be fully exploited on a single processor computer even with vectorization. Theoretically, a multi-processor computer with enough processors could update the E or H field accross the entire spatial grid in the time taken to update at one spatial point, providing of course that the computer had enough memory to simultaneously hold all field values in core (see, for example, the work by Davidson *etal* [9] on the implementation on the FDTD method using 8192 processors on a Connection Machine, model CM-2).

In practice, geometric decomposition is used to partition the grid into subspaces each of which is assigned to one processor. Due to the form of the finite difference equations, it is necessary to exchange data between processors, in a distributed or non-uniform memory access environment, at the end of each time step; this is needed in order to update field values along the interfaces. This corresponds, physically, to a travelling wave propagating across the total volume. In figure 1 we show the way in which we have partitioned the total volume among four processors; in general there may be N such processors. This is consistent with the linear topology of the 10 Base-2 thinwire ethernet (10 Mbits/second) which connects together the HP workstations in our laboratory.

In an attempt to obtain (static) load balancing and hence synchronization of the tasks we make sure that each processor is given an equal number of points to handle, meaning that the number of points along the Y-axis ($J_{\text{max}} + 1$) must be an exact multiple of the number of processors. This is a reasonable assumption for a *parallel virtual machine* consisting of a homogeneous network of *free* workstations. By *free*, we mean that no one is logged onto the workstation or a negligible amount of work is being performed, in other words the workstation is lightly loaded. The programs which update fields in the first and last regions (subspaces 1 and 4, respectively in figure 1) require boundary conditions to be obeyed but are otherwise identical to the program used in all other regions. The extra time consumed in applying the boundary conditions is insignificant compared to the time spent updating fields within the volume and does not therefore introduce a load imbalance into the virtual machine. These programs have been named, for obvious reasons, *first*, *last* and *middle*.

During a single timestep, a host must:

1. transmit/receive the E-field components from the previous time set-up (evaluated at the boundary between subspaces);

2. compute H-field values in the subspace;

3. transmit/receive the H-field components (in the opposite direction to the E-field values);

4. compute the E-field values in the subspace.

This process is continued until a steady state is achieved. The transmission and reception of boundary field values, at each timestep, is illustrated schematically in figure 2, for an arbitrary *middle* host called X. The *first* and *last* hosts also transmit and receive boundary field values but only half as many as a *middle* host. If figure 2 were redrawn for the *first* host, then the lefthand side would be missing, while a similar figure for the *last* host would omit the righthand side.

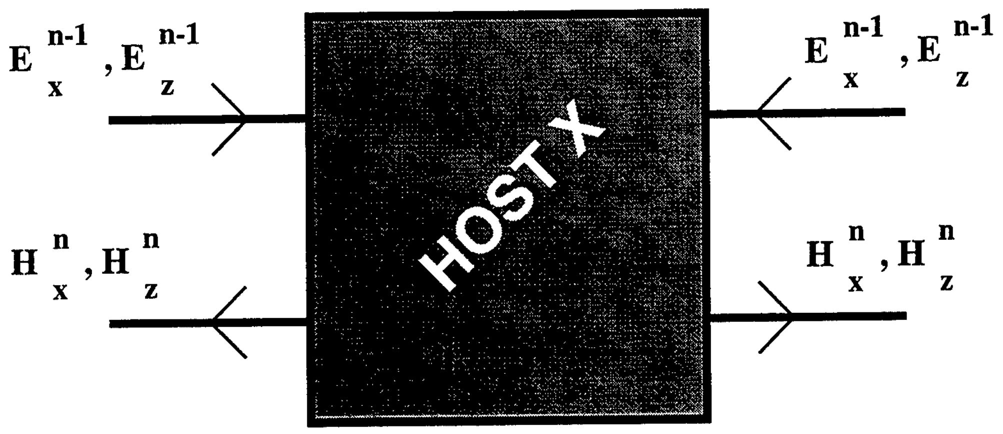


Figure 2: Communications of the electromagnetic field components for one host during the nth timestep; the geometric domain has been partitioned along the Y-axis only.

Message Passing Harnesses

Even today, many scientific programmers develop sequential codes in a top-down fashion with the Von-Neumann concepts of computer architecture in mind. Notwithstanding the fact that such codes are often restructured to take advantage of compiler optimizations exploiting hardware features such as pipelining, chaining and, perhaps, vectorization, the approach taken by scientific programmers has been largely to rely upon third party utilities/libraries to enhance their code performance; one example is the basic linear algebra system (BLAS). As long as these libraries are available on an architecture, codes dependent on the libraries are assured of high performance on that architecture. Consequently, in the era of parallel and distributed computing a number of *message passing harnesses* appeared obviating the need for scientific, particularly Fortran, programmers to learn about network programming. Message passing harnesses first appeared on transputers towards the late eighties, but with the adoption of the Internet Protocol (IP) as a defacto global standard in the early ninties harnesses quickly became available for UNIX workstations, and even UNIX mainframes, connected by IP networks.

The novice should appreciate that message passing harnesses are simply a layer of software that sit between an application program and operating system interfaces. It is the within the operating system that the physical transmission and reception of data takes place just as data is written to and read from disk. Sometimes the message passing layer is thin and provides very little abstract functionality, sometimes the converse is true. Related to the rapid growth in the use of message passing is the fact that the Berkeley Standard Unix distribution (BSD) implemented and standardized an abstract data construct known as a socket. In this model, the network hardware, the CPU hardware, their mutual interaction, the specifics of the protocol and the role of the operating system are all encapsulated within *socket* routines and thereby hidden from a socket programmer. This means that network programming becomes a matter of manipulating sockets and data buffers in real time, a task easily accomplished in the C language and frequently referred to as IP socket programming. With the standardization of the IP socket programming model, implementations of sockets have become available for all operating systems in common use. On a UNIX workstation, and on a PC, message passing harnesses are simply an interface between an application program and socket routines therefore.

In the late eighties and early ninties there was a proliferation of message passing harnesses, each providing different functionality and therefore incompatible. In the physical sciences and engineering communities PVM was seen as a de-facto standard and was, and continues to be, widely used. Many computer manufacturers now provide implementations of PVM which are highly optimized for use on their architecture just as they do also for BLAS routines. To date, there is no ANSI standard for message passing harnesses, however beginning in 1992 around forty organizations created a working group to propose, but not to implement as such, a 'standard' for message passing; this effort was known as the Message Passing Interface forum and the standard created is known as MPI. Vendors of massively parallel processors support the MPI standard on their own particular architectures. for example SGI/Cray on the Cray T3E; a number of implementations of MPI have also appeared for workstations, for example CHIMP: the common high level interface to message passing from the Edinburgh parallel computer centre[14]. In this article, we have chosen to focus only on PVM deferring consideration of other message passing architectures for future work.

Latency in PVM

The costs inherent in using PVM for our FDTD calculation are twofold, regardless of the computing environment used. Initially, there is a fixed set up cost due to the fact that the *first* process must spawn the *last* process and several copies of the *middle* processes and prepare them for subsequent FDTD timestep updates. This cost can be amortized over subseqent timesteps such that if there are very many timesteps, the setup costs become insignificant. The second, and critical, limitation is the cost of communicating boundary field values coupled to process synchronization, because this has to be done at every timestep and is therefore an increasing cost; in the event that this cost, within one timestep, exceeds the reduction in computation time for that timestep, derived from domain decomposition, then the distributed FDTD implementation will always require more time than its sequential analogue.

In simple terms, the time consumed in updating the electromagnetic fields in a subdomain depends on the total number of points in the volume and on the CPU speed; part of this time may be spent applying boundary conditions at the surface of the volume. Correspondingly, the time consumed in transferring boundary field values depends on the surface area of the volume and on the transmission time for a message between processes. Aside from the limitations of network hardware, the PVM system itself has an associated latency just as any message passing system does; the reader should appreciate that the design of a complicated utility such as PVM requires trade-offs between reliability, portability and efficiency. A rudimentary discussion of PVM, and a trivial implementation, can be found in Robbins and Robbins[15].

PVM uses a *spoke and hub* system akin to that employed in the airline transport industry. On each processor there is a PVM daemon (*pvmd*), that is a hub, which communicates directly with PVM tasks running on that processor using either TCP/IP sockets or Unix domain style sockets. The PVM daemons, on each processor, communicate with each other using UDP/IP sockets. A message from a task on host A destined for a task on host B, travels first to the pvmd on host A, then to the pvmd on host B and finally to the task on host B. Enroute there is some buffering through send and receive queues within the pvmds; more critically, the UDP/IP protocol is an unreliable delivery protocol requiring PVM to implement message fragmentation and an acknowledgement/retry mechanism to ensure message delivery. It must be pointed out that PVM does permit direct task to task communication using TCP/IP sockets and thereby avoiding the pvmds; this option is mentioned in the PVM user manual with the corollary that it does not scale well, therefore we have not used it in our present work.

Initial Results

The effectiveness of the parallel FDTD code was gauged by defining the speed-up, S, as the ratio of the execution times for the equivalent serial code (T_{ser}) to the parallel one (T_{par}), that is:

$$S = \frac{T_{\text{ser}}}{T_{\text{par}}}.$$

In the first instance, the network was not dedicated to these tasks. One can see, from figure 3, that the speed up for a given number of processors increases with the number of grid elements, that is with a decreasing mesh size, although there are some occasions where the speed-up drops below what one would expect. An example of this behaviour is the case of five processors with the finest mesh ($\delta = 1.48$ mm). The speedup

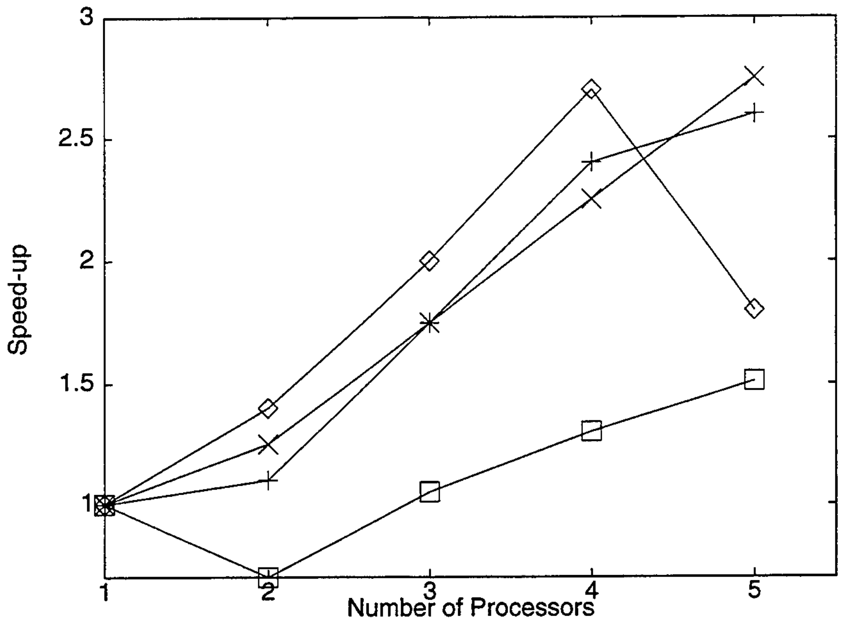


Figure 3: The speed-ups for up to five processors when the network was not dedicated to the PVM tasks. ◇ points are for a mesh size of 0.00148 metres, × for a mesh size of 0.0017 metres, + for a mesh size of 0.003 metres,□ for a mesh of 0.00616 metres.

would appear to far from optimal with $S = 1.86$. It was speculated that another user had begun a job on one of the hosts and so the run was repeated with the all workstations reserved for the duration of the test; results are shown in figure 4. For the particular case just mentioned, a much improved speedup, $S = 3.45$, was found. The approximate factor two increase in the speedup, that is a halving of the execution time, reinforces the assumption that two jobs sharing a host was the cause of the initial poor result. The speedup of $S = 3.45$ using five processors, with dedicated use of the network, is clearly the upper bound for this particular problem. We have shown that this is an unlikely figure to achieve under typical network conditions and have quantified the dramatic reduction in efficiency that can result from having just one host otherwise occupied. This is a striking reminder of the fact that desktop platforms, that is UNIX workstations and PCs, were never really designed as multi-user platforms to replace minicomputers or even mainframes. Unfortunately, faster CPU speeds have masked this problem somewhat. Simple economics combined with the fact that the operating systems in use on the desktop use premeptive multi-tasking has led, by default, to use of desktop platforms as minicomputers and even mainframe replacements however.

Surprisingly, another drop in efficiency was found, see figure 4, when the mesh corresponding to ($\delta = 1.70$ mm) was distributed over five processors. Following further checks, it was found that although six processors were configured, the PVM application takes one host to use as a group server when the programs include dynamic groups. Furthermore, the default host selection is through a *round robin* process which places a spawned task onto the next workstation in the host file. It was found that this method did not guarantee that each task was placed on a separate host giving rise to uncertainty in the obtainable speedup on a given run. Our method used the `PvmTaskDefault` option on the *pvmfspawn()* call, a feature that can be replaced by the `PvmTaskHost` option requiring a specific host to be given. The *pvmfconfig* and *pvmftasks()* routines provide the facility to interrogate the virtual machine and so discover, for example, which host is acting as the group server and which hosts are not yet used for PVM tasks. We could use these routines to make the code adapt to the tasks already present on the virtual machine, however we have not investigated this solution because it is encapsulated, by default, into the dynamic loadbalancing solution that we suggest in the next section.

It is noticable in figures 3 and 4 that the speedups for the three finest meshes, 0.00148, 0.0017 and 0.003 metres, are clustered together and well separated from the data for the most coarse mesh, 0.00616. We

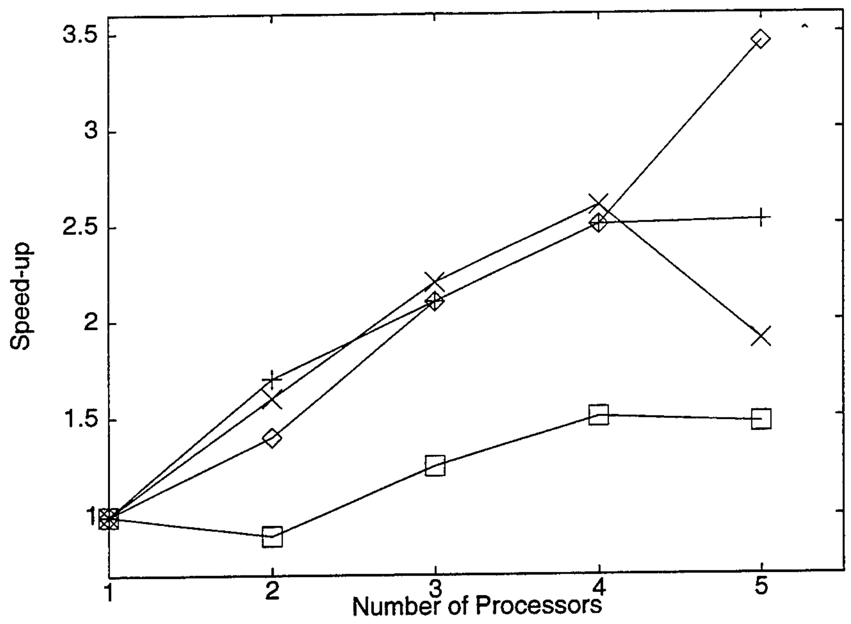


Figure 4: The speed-ups for up to five processors when the network was dedicated to only the PVM tasks. As in figure 3, ◇ points are for a mesh size of 0.00148 metres, × for a mesh size of 0.0017 metres, + for a mesh size of 0.003 metres,□ for a mesh of 0.00616 metres.

performed a calculation for the intermediate mesh size of 0.00403 metres and found that the speedup values clustered with the 0.00616 metre case. The overall trend is, as mentioned above, that a finer mesh benefits more substantially from concurrent computations however the increase is not linear. We believe that this reflects performance characteristics of the data buffering that is part of the any message passing process, coupled perhaps to the paging of virtual memory by the operating system. An investigation of the buffering issue could be achieved by using small RISC assembler routines to write those sections of the code concerned with moving blocks of field data around; unfortunately, we have have not had time, or resources to do this as yet.

For problems which require a large number of geometric grid points, the memory requirements can often be a limiting factor in performing an FDTD computation; under these conditions optimum CPU performance is a only secondary issue relative to actually solving the problem at all. On a single processor, or even on a massively parallel processor, the addition of extra memory to the system is generally not an available option for a user. Distributed computing has the distinct advantage that to obtain more memory one need only find a lightly loaded, networked workstation, or a networked PC, within one's organization. Typically, there are many such machines available, even if only during the night and at weekends.

Improving the Effectiveness of the Computation

In order to improve the predicability of execution times of the distributed algorithm, we decided to assess the number of executing processes on each host using the well known UNIX command *uptime*[16], which provides a one line summary of the number of users and the average load on the processor for the previous five, ten and fifteen minutes; unfortunately, this approach proved to be ineffective in our circumstances. Some workstations in our laboratory act as file servers holding various commercial EM simulation packages; even when a simulation is being performed on another machine, the file server consumes CPU cycles and appears loaded. A more useful UNIX command is *top* which tabulates running processes and gives the percentage CPU consumption associated with each. This, of course, produces several lines of output which must be analysed to find the appropriate information, a task that was accomplished by using a UNIX C shell script.

This script was successful in selecting the lightly loaded hosts prior to FDTD execution. In addition to configuring the optimum virtual machine, the script compiled and ran the program on the available hosts. The code was modified so that each task was spawned onto the next host in the configuration so that all available hosts were used even though we use dynamic groups. This ensured that one, and only one, task was given to each workstation and thus helped to ensure that the impairment in effectiveness, described in the previous section, was eliminated. This improvement is of limited use, however, because in the duration of execution, the load on each of the networked workstations is likely to change due to the presence of other users. It was imperative to allow for the transient nature of the load, so methods of combating the negative effects of a multi-user system were considered.

The power of a portable parallel environment such as PVM lies in the fact that computational resources can be utilised which would be otherwise idle. However, to convince the owner of a private workstation that it should become part of a larger, virtual machine it is often necessary to ensure dedicated use of that host when the user requires it. It was with this scenario in mind that the following task migration scheme was developed. The migration of a complete task from a heavily loaded host onto a lightly loaded one is a more severe option than obtaining load balancing through load redistribution but it is easier to achieve.

In the virtual machine one has, in general, one incidence of the 'first' and one incidence of the 'last' program executing but several incidences of the 'middle' program executing. In order to simplify matters, only the incidences of the 'middle' program were condidered as candidates for migration. To enable the scheme, a fourth program source, 'middle-migrated' was created; this was designed a little differently from the 'middle' program so that it could receive data defining the calculation in the midst of a loop over timesteps and then continue executing from some arbitrary timestep.

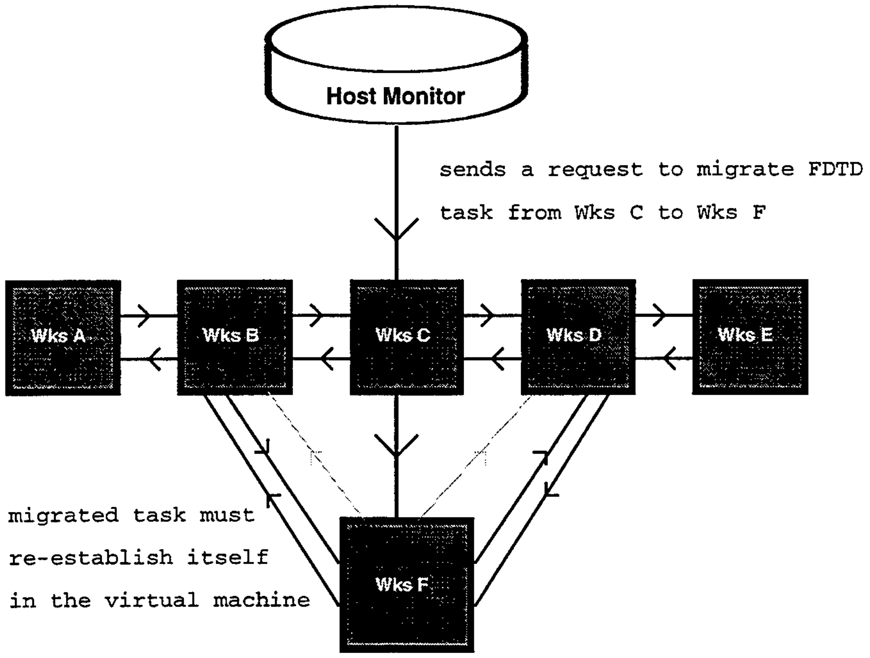

Host Monitor
sends a request to migrate FDTD task from Wks C to Wks F
Wks A
Wks B
Wks C
Wks D
Wks E
Wks F
migrated task must re-establish itself in the virtual machine

Figure 5: Schematic illustration of the communications involved in a task migration from workstation C to F. F must establish itself with workstations B and D as a nearest neighbour node.

At the start of an FDTD calculation, 'first' FDTD program spawns a *monitor* program after the FDTD calculation has been started, that is after some appropriate number of timesteps (defined by the user). The monitor then runs independently of the FDTD codes and continually assesses the status of each configured host. The monitor program is itself a master process controlling several slaves. In this case the slaves are

UNIX C-shell scripts which are spawned on each workstation in the virtual machine in turn. The scripts execute the *top* command with the output being directed to a file; this file is edited, using *sed* amd *awk* within the script finally producing a disk file which contains some integer data related to the load on the processor. This data is subsequently read from the file by the *monitor* program, a feature that depends on the network file system (NFS) that is in use in our laboratory (using NFS means essentially that all workstations share a common disk area). The efficiency of using NFS as a data sharing mechanism has been illustrated for image processing calculations using a network of SUN SPARC10 workstations[17].

A task migration is effected when the monitor program detects an overloaded machine and prompts the offending task to relocate on a host deemed to be free. The decision criteria programmed was that

> when two jobs are running on a host and one is an FDTD executable, then the FDTD process should be migrated off that host.

This was moderated by the condition that migration was not carried out until an FDTD executable was found to be in need of migration on three sucessive occasions. This condition was needed in order to prevent spurious migrations due to very short lived processes that might be present in the virtual machine. In figure 5 we show, the sequence of events that takes place when a task migration is effected.

In a non-migrating FDTD calculation, the communications among the various tasks are synchronous. In the load balancing situation several communications become asynchronous in nature, for example,

- between the monitor and the 'middle' programs.
- between a terminating task and the *migrated* one that replaces it.
- at the termination of the monitor program, something which takes place when the 'first' program reaches the end of the loop over timesteps, a message is passed asynchronously from the 'first'program to the 'monitor'program.

We have enabled these communications with the *pvmfprobe()* routine. A key issue was whether, or not, the extra probes and barriers used in the task migration scheme actually affected execution times. At first a probe and a barrier were used on each time step as it was felt that for computationally intensive FDTD applications these would have negligible effect, in other words the time spent updating electromagnetic fields was far in excess of that used for task synchronization. For coarse grids, or substantially faster CPUs, this would not be the case but then the distributed algorithm is ineffective for those anyway. The migrating and non-migrating FDTD codes were run for fifty timesteps with a selection of grid sizes; the execution times are compared in table 1. As expected, the more the computation involved in each time step, the smaller

Grid spacing (δ) in metres	Migrating Code	Non-migrating Code	Increase in the execution time
0.00616	5.82	2.12	2.74
0.00198	41.47	21.29	1.94
0.00148	88.66	54.18	1.64

Table 1: Comparison of execution times for the migrating and non-migrating FDTD codes, discussed in the text, for various sizes of mesh. All times are given in seconds.

the effect of the probes. The effect is still excessive for the finest mesh and this would put into doubt the advantage of parallelizing the FDTD algorithm at all. It is clear, upon reflection, that probing on each timestep is overkill and every ten steps would be a much more reasonable approach. Initial tests show that this produces an efficient task migration procedure.

Conclusions

The performance of our distributed implementation of the FDTD algorithm has been found to be sensitive to the dynamic load on the *virtual machine* used. The presence of other active tasks on any one, or more, of the nodes used causes severe degradation in performance. To overcome this limitation, a dynamic loadbalancing strategy is used in which tasks migrate onto lightly loaded nodes. Notwithstanding any performance degradation, distributed computing provides an easily scalable memory environment, a feature which is generally not economic on massively parallel systems.

The full task migration process described in this paper is effective when enough workstations are available so that a free machine can be found. It is not so useful for a small number of host machines when it is likely that all available hosts will be employed at the start of an FDTD execution. A more useful approach in these conditions is to use data redistribution. It is envisaged that the above programs could be modified so that instead of ruling out a host with one non-FDTD process running, we could steal half of its available CPU. Thus a free machine would calculate x nodes while a semi-free machine would compute $x/2$ to achieve a load balance on the homogeneous network. The dynamic load-balancing would thus involve a reshuffling of the amount of work on all machines when one has to be altered. At first sight, this seems so much more involved than full task migration to be unfeasible. However, the large increase in work needed by the monitor will not be a problem as it can run virtually independent from the FDTD codes. The number of communications during a load redistribution is undoubtedly larger than in a full task migration. These data transfers will occur concurrently and thus be quicker. It should therefore be a relatively simple matter to extend the task migration to produce a very efficient load balancing tool. This would not be limited to a homogeneous network because it is based on standard UNIX and PVM commands.

Acknowledgments

This work was supported, in part, by the UK Engineering and Physical Sciences Research Council under contract GR/L23215.

References

[1] Yee K S 1966, "Numerical Solution of Initial Boundary-value Problems Involving Maxwell's equations in isotropic Media ", IEE Trans Ant. Prog. Vol: AP-14, May 1966 pp.302-307.

[2] Huang T W, Houshmand B and Itoh T 1993, "Microwave structure characterization by a combination of FDTD and system identification methods", *IEEE MTT-S Int. Microwave Symposium Digest*, vol. 2, pp. 793-796.

[3] Kunz K S and Lee K M 1978, "A three-dimensional finite-difference solution of the response of an aircraft to a complex transient EM environment I: The method and its implementation", *IEEE Trans. on Electromagn. Compat.* vol. 20, pp. 328-333,

[4] Chebolu S, Mittra R and Becker W D 1996, "The analysis of microstrip antennas using the FDTD method", *Microwave*, vol. 39, pp. 134-150.

[5] Ghandi O P, Sullivan D M and Taflove A 1988, "Use of the finite-difference time-domain method in calculating EM absorption in man models", *IEEE Trans. Biomedical Engineering* vol. 35, pp. 179-186

[6] Buchanan W J 1993, "Simulation of radiatin from a microstrip antenna using three dimensional finite-difference time-domain (FDTD) method." *Eighth International Conference on Antennas and Propagation*, IEE Conf. Pub. No. 370, pp639-642

[7] Chew K C and Fusco V F 1995 "A Parallel Implementation of the Finite Difference Time Domain Algorithm"*Intl. J. of Numerical Modelling:El. Networks, Devices and Fields* 8 293

[8] Taflove A 1995 *Computational Electrodynamics: The Finite Difference Time Domain Method*, (Boston: Artech House) ISBN 0-89006-792-9, pp545-84

[9] Davidson D B, Ziolkowski R W and Judkins J B 1994, "FDTD modelling of 3D optical pulse propagation in linear and non-linear materials", *Second International Conference on Computation in Electromagnetics*, IEE Conf. Pub. No. 384, pp166-169

[10] Varadarajan V and Mittra R 1995 "Finite Difference Time Domain (FDTD) Analysis using Distributed Computing" *IEEE Microwave and Guided Wave Letters* 4 144-145

[11] Rodohan D P, Saunders S R and Glover R J 1995 "A Distributed Implementation of the Finite Difference Time-Domain (FDTD) Method" *Intl. J. of Numerical Modelling:El. Networks, Devices and Fields* 8 283

[12] Reale F, Bocchino F and Sciortino S, "Parallel Computing on UNIX Workstation Arrays" *Comp. Phys. Comm.* **83** pp. 130-140, 1994

[13] Geist G A, Beguelin A, Dongarra J, Jiang W, Manchek R and Sunderam V 1994, *PVM 3 User's Guide and Reference Manual*, Oak Ridge National Laboratory Technical Report ORNL/TM-11616.

[14] Malard J 1995, *MPI: A Message Passing Interface standard*, Technical Report, Edinburgh Parallel Computer Centre.

[15] Robbins K A and Robbins S 1996, *Practical Unix Programming: A Guide to Concurency, Communication and Multithreading*, (Prentice Hall: NJ) ISBN 0-13-443706-3 p401-18.

[16] "Using HP-UX", HP 9000 Workstations, Hewlett Packard, Part No. B2910-9001, Edition 1, 1994.

[17] Ward G, "Parallel Rendering on the ICSD SPARC-10's"
The Radiance Synthetic Imaging System, http://radsite.lbl.gov/radiance/refer/Notes/parallel.html.

PARALLEL COMPUTATION OF LARGE-SCALE ELECTROMAGNETIC FIELD DISTRIBUTIONS

Peter S Excell, Adam D Tinniswood* and Kathleen Haigh-Hutchinson
Department of Electronic and Electrical Engineering, University of Bradford, Bradford, UK
*Now with Department of Electrical Engineering, University of Utah, Salt Lake City, USA

ABSTRACT. Some experience of the use of high-frequency electromagnetics software on parallel computers is reported. Types of such computers are reviewed and approaches to the parallelisation of existing serial software are discussed. A practical large-scale problem is presented, involving the modelling in very fine detail of electromagnetic penetration into biological systems. This was tested on state-of-the-art parallel computers and important practical and strategic aspects of the experience derived are discussed. It was found that considerable programmer effort was required to optimise the software to use the computer architecture effectively, but that efficient acceleration of the run-times of typical computational tasks could be achieved, provided that the tasks were large and were partitioned optimally.

1 INTRODUCTION

The computation of electromagnetic field distributions almost invariably involves the repetition of a similar set of computational actions for a large number of data points. For systems of practical interest, this implies a very large computational task. Although it is possible to construct certain problems in a way that enables them to be run on computers of modest power, structures which are large in comparison with the wavelength frequently require the use of the most powerful computers available, usually characterised by the name 'supercomputers'.

In recent years, some of the problems that stretch such supercomputers to the limits of their abilities have been designated as 'grand challenges'. Lists of such grand challenges tend to concentrate on applications in pure science and environmental modelling, such as high-energy physics, quantum chemistry and atmospheric circulation modelling. Computational fluid dynamics (CFD) is also frequently included in the lists, but other applications of industrial interest tend to be given less emphasis. Although it is relevant both to science and to industrial applications, it is frustrating to find that electromagnetics does not appear to have been explicitly mentioned in any well-publicised list of 'grand challenges' and it is very desirable that this situation be remedied.

The main exercise used to test both hardware and software in the present work was the modelling of electromagnetic wave penetration into the human head, using a head image classified from a magnetic-resonance image (MRI), standardised to 1 mm resolution, which gives about 10^7 FDTD nodes [1] (Fig. 1). This is a particularly challenging test due to the discontinuity between air and the very large permittivity of living tissue. The source transmitter was a representation of a generic mobile telephone handset, which was essentially a plastic coated box, in the centre of the top of which was a quarter-wave wire antenna driven at its base with a continuous wave signal. The two frequencies of interest were 900 MHz and 1800 MHz, which are used as the carrier frequencies for most mobile telephone technology in Europe. Although the model of the head has been classified to a resolution of 1 mm it was also re-sampled to resolutions of 2 mm and 2.5 mm to allow the simulation to be run in serial form on a UNIX workstation (Sun Sparc 20 with 128 MBytes of RAM), and to facilitate comparisons of different parallel processing systems. The majority of this work was undertaken as a contribution to a broadly-based European Union project (EUROPORT-2) which aimed to demonstrate good scaleability performance and ease of inter-platform portability for several examples of industry-standard software ported to a range of contemporary parallel computers.

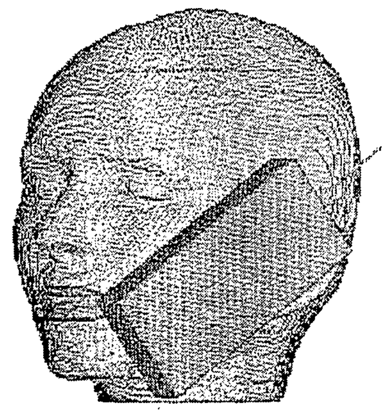


Fig. 1 The original (1 mm resolution) head dataset, used as a test piece. Shown here with the generic mobile telephone: the version used in Section 4.2 included a simulated hand.

2 PARALLEL PROCESSING SYSTEMS

The idea of sharing a computational task between many processors in order to achieve higher speed or larger task capacity has been appreciated from the early days of computing. Unfortunately, it often proved impossible to achieve an adequate performance with such systems, except in a few specialised cases, usually using application-specific hardware. The manufacturers of the most powerful hardware thus concentrated on efforts to increase the speed of traditional serial processors. An important hybrid technique that evolved was 'pipelining', in which identical operations were undertaken on several separate data elements in rapid succession.

The traditional serial processor is categorised as 'single-instruction single data' or SISD and the pipeline processor as 'single-instruction multiple data' or SIMD. Computers employing pipeline processors are usually known as 'vector processors': they have some of the characteristics of parallel processing since the pipelined serial operations are almost indistinguishable from parallel operations. Clearly, independent parallel processors can also function in 'multiple-instruction multiple data' or MIMD mode, but it is rare for different processors to undertake completely heterogeneous tasks as it is extremely difficult to distribute such tasks efficiently between processors. With the increasing power of individual processors it has often become irrelevant to consider tasks at the 'instruction' level since a marginal amount of loss of strict synchronism may mean that processors are not executing similar instructions precisely simultaneously. Taking a broader view, it is more relevant to compare the program segments that each processor is running and hence the abbreviations SPMD and MPMD are now also widely used, where 'P' stands for 'Program'.

By the mid-1980's, it appeared that the vector-processing architecture was approaching the limit of its ability, and hence vector processor manufacturers started to incorporate parallel processing into their computers, but only to a limited degree (typically two to sixteen processors, exemplified by the Cray X-MP and Y-MP computers). Smaller manufacturers had continued to develop parallel computers, many exploiting the Transputer, a microprocessor optimised for parallel applications, but their success was limited, both technically and financially. At least two manufacturers (Thinking Machines and Active Memory Technology) took an extreme path of developing systems with very large numbers (>1000) of low-power processors, but this approach did not achieve widespread success [2]. The 1990's have seen a revival of interest in computers with a modestly large number of processors (typically between 16 and 512 at present) and this type has at last become accepted as the mainstream supercomputer architecture. A re-evaluation of designs with several thousand processors also now appears to be developing, based on low-cost personal computer architecture.

Within the class of parallel computers, a new subdivision emerged, between those having 'shared memory', a single large data store used by all of the processors, and those having 'distributed memory', i.e. standard memory modules attached to each processor. The shared memory approach appeared more natural to program, but it suffered from problems of low speed of access by the processors and complexity in the hardware arrangements. The distributed memory approach enabled a still greater use of standard hardware, but required great care in programming since time could be wasted searching for an item of data in another processor's memory. Alternatively, two versions of the same data could exist in the memory of different processors and there could then be uncertainty over which was the valid version. In practice, the hardware arguments favouring distributed memory have become very strong, and the software techniques required to exploit it are slowly developing towards maturity. Nonetheless, powerful shared-memory machines continue to be made, and their performance is again becoming competitive with distributed-memory machines.

In recent years, another form of distributed-memory parallel processing has appeared as a result of the installation of large networks of UNIX workstations in many companies and universities. These workstations spend much of their time idle, particularly at night, and hence techniques have been devised to take control of unused power in a network and distribute a program across a large number of workstations, the results being fed back to a designated master processor. This form of parallel processing suffers from relatively poor communications, and hence it is important to partition the task in such a way that the communications overhead is minimised. On the other hand, the processing power is available at minimal cost, and hence it has been an attractive option for many organisations [3].

3 PORTING OF LEGACY SOFTWARE TO PARALLEL PLATFORMS

In earlier years, when attempts were made to transfer (or 'port') electromagnetics software from traditional serial machines to multi-processor machines, it was frequently found that some speed-up was achieved when using a few processors in parallel, but when larger numbers were tried the software would not run with the expected speed increase, and it was not unknown for it to run even more slowly than in its serial form on a single processor. The number of processors at which this problem arose could be as small as four [4], and 'Amdahl's Law' was frequently invoked to explain the effect

[5, 6]. This law basically states that there is a limit to the amount of speed-up that is achievable with parallel processing, because real-world software always contains some serial sections which rapidly come to dominate the run time when the parallelisable sections have been accelerated. In practice, the situation is even worse than Amdahl's Law suggests, because parallel sections of the program require extra time for setting up and distribution of the data between the designated processors. When running, the parallel sections can be severely handicapped by communications delays in passing intermediate data between processors, especially if the algorithm is not optimised to minimise communications requirements.

4 PARALLELISATION OF SOFTWARE

Most single-box parallel computers have been sold with accompanying software which attempts to translate existing serial programs in a standard language (usually Fortran) into a form which exploits the parallel architecture efficiently. Similar software has also been provided by vector processor manufacturers, but for both architectures it has been found that the automatic vectorisation or parallelisation routines frequently make poor decisions on the re-working of a serial code, or fail to notice a section of code which is capable of modification. 'Manual' parallelisation or vectorisation has thus usually been used, a highly-skilled programmer making appropriate modifications to the program based on a deep understanding of the architecture of the computer and the operation of the program.

It seems intuitively reasonable to suggest that this situation should not persist in the long term and that fully reliable parallelisation software should eventually become available. This would be given information on the architecture of the machine on which it is running, would then deduce the structure of the serial program that it had been given and thus devise an optimum strategy for partitioning this onto the parallel system. This problem is the subject of several international initiatives which are reviewed below.

Fortran 90 incorporated intrinsic matrix manipulation functions, constituting some of its most significant extensions over Fortran 77. Unfortunately, the efficient execution of such matrix operations is very dependent on the structure and content of the matrices involved and the detailed architecture of the computer in use. To overcome these problems, extensions to Fortran 90 have already been adopted by users of parallel processors and this extended language became known as High Performance Fortran (HPF), which is now strongly influencing the emerging Fortran 95 standard.

In attempts to address the particularly difficult but important problem of efficient partitioning of an algorithm across a network of UNIX workstations, some new approaches were devised. The most important of these were PVM (Parallel Virtual Machine) and MPI (Message-Passing Interface). Although these methods were generally devised for use with asynchronous networks of workstations, they have been adopted equally widely with single-box parallel supercomputers. PVM was initially the most popular method: this loads 'spawned' replicated versions of the main program into each of the designated slave processors, which then proceed to function autonomously [7]. The efficient partitioning of the problem between processors is still under the control of the programmer, although separate graphical performance-indicating tools are available [8] to give an indication of the way in which the program is functioning, the key objectives being 'load balancing' between the processors, and minimisation of the communication activity. MPI was defined later than PVM, but it has now achieved greater popularity. It functions in a similar way to PVM but does not use the organic 'spawning' approach.

Early experience was gained in efforts to vectorise and parallelise standard Method-of-Moments (MoM) codes, with some success [9]. It was found particularly difficult to vectorise and parallelise the critical matrix-solution phase of MoM because of the inherently high degree of interaction between the elements of the very dense interaction matrix that is needed in this method. This means that there is a high communication overhead if the matrix elements are distributed across a parallel processing system. There is also 'data dependency' with both parallel and vector processors, forcing some parts of the process to be handled in a sequential way in order to ensure that the correct version of the input parameters is being used. Filling of the matrix and computation of the final field distributions do not have the same difficulties and these operations can be parallelised relatively easily.

In considering this experience, it became apparent that a method based on a differential-equation formulation, such as the Finite-Difference Time-Domain (FDTD) method, would have an inherent advantage on a distributed (parallel) processing system. This is because the interaction matrix is effectively sparse and the updating of an individual node only requires data from its immediate neighbours, whereas nodes in an integral-equation formulation (e.g. MoM) require data from all of the other nodes in the system. This effect is closely similar to the dichotomy between the competing physical approaches of 'action at a distance' (c.f. integral-equation formulations) and 'fields' (c.f. differential-equation formulations).

A very significant overview of techniques for parallelisation and vectorisation of FDTD algorithms was presented by Gedney and Barnard [10], in particular discussing methods for maximisation of the efficiency of DO-loops, which are the main parallelisable component of programs written in a traditional language.

4.1 Experience with the FDTD Method on a Parallel Computer with Virtual Shared Memory

The KSR-1 was a novel design of parallel processor, produced by the Kendall Square Research Company in the USA. Although the company has now withdrawn from the high-performance computing market, the machine used an interesting architecture which may influence future designs and hence experience gained with it is still relevant. The particular machine used had 64 processors, although the maximum number that could be allocated to a single job was 48; the standard word length was 64 bits. The machine had a unique 'virtual shared memory' architecture, in which sets of processors were connected in hierarchical rings, each having fast communications. The objective was to give an approximation to the behaviour of a shared-memory parallel processing system, whilst using the cheaper, standard, hardware of a distributed memory system. A cache was interposed between each local processor memory and the communications ring to accelerate the accessing of data by other processors.

The Finite-Difference Time-Domain (FDTD) method for electromagnetic field computation was used to test the ability of this computer, the program used being an updated version of THREDE [11]. The test exercise was the modelling of electromagnetic wave penetration into the human head, as discussed in Section 1, using the head image re-sampled to 2.5 mm resolution, which gives about 10^6 FDTD nodes.

In converting the program for parallel operation, the computer manufacturer's automatic parallelisation tool was first tried, but this only gave significant benefit on the most trivial DO-loops (this experience was also found with similar tools on other computers). The virtual shared memory was also found to be a potential source of problems, much like virtual memory on earlier serial systems, in which inefficient page swapping could slow down program execution dramatically if the task was not correctly partitioned. Data was normally swapped between processors in 'subpages' and it was found that maximum efficiency was achieved in the FDTD algorithm when the field data arrays were each arranged to start on a subpage boundary. It was concluded that, for satisfactory performance, memory must be managed explicitly by the programmer and internal automatic procedures cannot be relied upon. Similarly, it was found that the automatic procedure frequently decided to use fewer than the maximum number of processors for many sections of parallelised code: explicit control of the number of processors thus also appears essential. Explicit 'affinity directives' were thus used to instruct the operating system to partition the arrays in a pre-determined pattern, and to use an identical pattern for all of the arrays: this enabled remote data to be located rapidly, rather than time being wasted in a search. It was necessary to use explicit correlation of the data partitions and the number of processors, otherwise the default algorithm in the operating system would tend to make non-optimum, uncorrelated, decisions on these parameters. To avoid re-compilation when the number of processors was changed, this number was configured as a system variable, to be set outside the program.

Extracting optimum performance from a machine of this type invariably involves a number of minor code modifications, based on experience and a detailed understanding of the operation of the hardware and software. In this case, a problem was found in that the existing code, optimised for a vector processor, used separate loops to iterate the FDTD time-marching equations in the x, y and z planes. When this was altered to a single loop to update all components at once for each cell, the amount of data loading and storing required was reduced by a factor of three (this is discussed more fully in [10]). Loading and storing is a slow process on this computer, whereas it is fast on a vector processor. This modification gave a speed improvement by a factor greater than ten. It was found that the computer was particularly fast with multiply operations but slow with division, load, store and trigonometric functions. To circumvent this problem, the number of divisions was reduced by converting commonly-used denominators into reciprocals which could then be used with the multiply function. Similarly, commonly-used trigonometric function results were stored to avoid fresh function calls.

4.1.1 Scaleability

'Scaleability' is an important objective of parallel computers and software. A hardware-software combination has perfect scaleability if the program runs N times faster when the number of processors in use is increased by a factor N. In practice this never happens, but a viable system can be expected to approach at least about 50% of this ideal performance. Popular measures of parallelism performance are 'speed-up' and 'parallelism efficiency', which are defined thus:

Speed-up $$S = \frac{T_1}{T_n} \qquad (1)$$

$$\text{Parallelism efficiency } \eta = \frac{T_1}{nT_n} \times 100\% = \frac{S}{n} \times 100\% \quad (2)$$

where T_1 is the execution time on one processor and T_n is the execution time on *n* processors.

The product of the number of processors and the observed run time can also be used to give an indication of the parallelism efficiency when T_1 is not known. This product is clearly equal to T_1 when the efficiency is 100% and will be greater if the efficiency falls.

As a test of the scaleability of the modified THREDE program, the 2.5 mm resolution dataset was run for 250 iterations (not enough for convergence, but adequate for this test), on 10, 20 and 30 KSR-1 processors. Results are shown in Table 1: near-linear scaleability was found from 10 to 20 processors, but a substantial loss of efficiency occurred with more, probably due to an excessive communications burden resulting from non-optimum partitioning of the data matrix. The scaleability and speed-up should strictly be calculated using the run-time with just one processor as the baseline. However, the task was too large to run on one processor, as is very often the case. The result is that scaleability has to be assessed from the results with a larger number of processors (ten in this case), under the assumption that near-perfect performance is achieved up to this number. An alternative performance measure that has been used in some circumstances is 'scaled speed-up' [12], in which the size of the computational task is scaled in proportion to the number of processors. This gives results that appear better, but it was impractical to scale the task studied here down to the level that could be run on one processor.

Table 1. Scaleability tests with KSR-1 computer (One-dimensional partition; 100×100×100 nodes; 250 iterations)

No. of Processors	Run time (mins)	Processors × Time (mins)	Speed-up (see note)	Efficiency (see note)
10	20	200	10	100%
20	10	200	20	100%
30	15	450	13.3	44%

Note: task too large for one processor, hence speed-up and efficiency calculated by assuming they are ideal with 10 processors.

4.1.2 Partitioning of the Problem Space

A diagrammatic representation of the way in which the problem space was partitioned on the KSR-1 is shown in Fig. 2. The problem space was divided into a number of slices, each to be computed on a separate processor (for simplicity, only five slices are shown). This is a one-dimensional partition, but two- and three-dimensional variants are also possible (see below). It is important to give each processor approximately the same workload, as the computational effort is almost identical in each node of the FDTD calculation. An unbalance in workload leads to wasted processing time on idle processors, thus reducing efficiency. Since the 2.5 mm dataset had array dimensions of about 100 in each direction, thirty processors would each only be handling three or four slices of the dataset, and a proportionately increased number of data transfers would be required to access the field data from adjacent processors. Even if the amount of serial code in a program were to be zero, it is thus evident that a point would always be reached where a problem of a given size would start to run more slowly as the number of processors was increased, since communications activity would start to dominate over computation. This is true of any parallel processing architecture, not just the rather unusual case of the KSR-1.

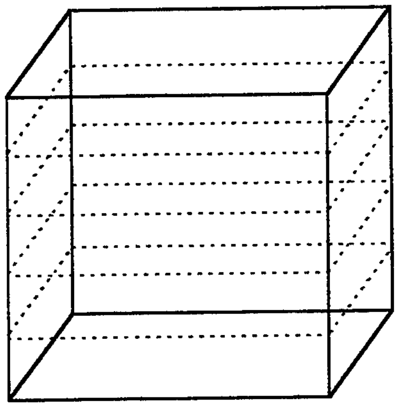


Figure 2: A one-dimensional partitioning of the FDTD problem space into sub-volumes, each to be handled by a separate processor.

A three-dimensional partition is illustrated in Fig. 3: here the problem space is shown partitioned for execution on a hypothetical eight-processor computer. Each of the eight processors is assigned an equal part of the problem in such a way that inter-processor communications are minimised.

Treating these representations of a three-dimensional data array as fictitious three-dimensional solids, it can be said that the 'surface area' of any sub-volume is proportional to the sum of the amount of inter-processor communication it will have to undertake, plus the amount of absorbing boundary condition (ABC) treatment it will need. The latter only applies to the outer surfaces of sub-volumes at the boundaries, but this type of surface will be in a majority for regularly partitioned cubic volumes having fewer than 12 sub-volumes. The 'volume' of any sub-volume is proportional to the amount of internal calculation it will have to undertake, using data stored in its own memory. ABC treatments may be expected to be a little slower than standard FDTD calculations, but communications are almost invariably very significantly slower. Thus the optimum strategy should be to aim for partition sub-volumes that have a maximised ratio of volume to surface area.

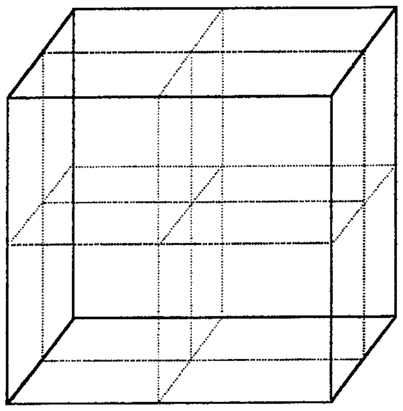


Figure 3: A three-dimensional partition of the FDTD problem space.

Using the one-dimensional partition, the total surface area of all of the sub-volumes, measured in data elements, is:

$$A_1 = 2n^2(p+2) \qquad (3)$$

where n is the number of data elements in each dimension of the problem space and p is the number of sub-volumes (i.e. processors). The factor 2 is included because each internal surface is seen twice, once from each side. The problem space is assumed to be a regular cube for simplicity: the same general conclusions will be valid for cuboidal volumes except when they are very long and thin.

The total *communicating* surface area of all of the sub-volumes (i.e. excluding external surfaces requiring ABC treatment) is:

$$A_{C1} = 2n^2(p-1) \qquad (4)$$

However, for the three-dimensional partition the equivalent total area is:

$$A_3 = 6n^2\sqrt[3]{p} \qquad (5)$$

And the communicating area is:

$$A_{C3} = 6n^2(\sqrt[3]{p}-1) \qquad (6)$$

since there are $\sqrt[3]{p}-1$ internal planes in each of three co-ordinate directions, each seen from two sides and each with area n^2.

Taking, for example, the perfect cubes p = 8, 27 and 1000, these equations give the results shown in Table 2.

Table 2: Typical total areas of sub-volume surfaces, from Eqns (3) to (6)

p	$\sqrt[3]{p}$	A_1	A_{C1}	A_3	A_{C3}
8	2	$20n^2$	$14n^2$	$12n^2$	$6n^2$
27	3	$58n^2$	$52n^2$	$18n^2$	$12n^2$
1000	10	$2004n^2$	$1998n^2$	$60n^2$	$54n^2$

Since the volumes of the sub-volumes are constant in each row of this table (volume = n^3/p), these results show that, whether or not ABC treatments are considered to have a significant effect on the size of the task, the ratio of 'surface area' to 'volume' for each processor will be much smaller for three-dimensional 'cubic' partitions than for one-dimensional 'slice' partitions.

This shows that the 'slice' partition used in the KSR-1 experiments was non-optimal, and it explains the poor efficiency obtained with 30 processors (it was not possible to repeat the experiment with cubic partition as support for the computer was terminated following the untimely withdrawal of the manufacturer from the parallel computer business). The KSR 'virtual shared memory' architecture is not being perpetuated at present, although it has some similarities with that used in another current manufacturer's products. Nonetheless, experience gained with such experimental architectures is always instructive in assessing the viability of future architecture proposals.

It should be noted that the above calculations only give an indication of the relative amount of time required for communication of data when comparing the same problem run on different topologies (an estimate of communication time relative to computation time is given in [13]). The basic fact remains, however, that a calculation that involves data communicated between processors will be slower than an equivalent one that does not. It therefore appears that the optimum strategy will always be to minimise the ratio of communication to internal processing tasks, per processor.

4.2 Experience with Parallelised FDTD using PVM.

In 1994-95 the European Union's ESPRIT III action initiated a programme entitled EUROPORT, the object of which was to demonstrate portable and scaleable parallelisation of industry-standard programs, using automated tools as far as possible. 'Portable' means that the program will operate, without substantial modification, on a range of platforms, including parallel processors from various manufacturers, and workstation clusters. One of the projects was PEPSE (Parallel Electromagnetics Programming Support Environment) [14], which was concerned with parallelisation of a standard FDTD program and a linked graphical input-output program.

The FDTD program chosen for parallelisation was EMA3D, which is similar to, and has evolved from, THREDE [11]. The parallelisation was undertaken using PVM [7], due to its widespread acceptance in the parallel-processing community at the start of the project. This operates by running a supervisory 'C' program which calls appropriate PVM library routines. These 'spawn' copies of the main task program (which may be in Fortran, but called from a C program) onto designated processors. The partitioning of the computational task is controlled by passing variables to the replicated programs, chosen to control the portion of the data that is to be associated with each processor.

The main time-marching FDTD equations and the ABC treatments were fully parallelised to ensure efficient execution of the code. However, the thin wire algorithm used for the handset antenna [15] was parallelised in a degenerate manner. This means that the thin wire computation was carried out on the root processor (the one assigned for supervisory code and serial parts of the program), and the results of this distributed to the relevant processors. Many of the problems used as test cases for the parallel FDTD code make use of thin wire sections in the code and could be slowed down by this. It is not, however, a computationally significant part of the modelling of these problems and thus it does not greatly affect the efficiency of the parallel code.

A small scaleability test, involving the modelling of simple dipole radiation, was undertaken with the FDTD software on two Meiko CS-2 parallel computers: the size of the problem space was $120\times240\times50$ ($=1.44\times10^6$) nodes, run for 500 iterations. Results are shown in Table 3: the 'topology' indicates the way in which the x, y and z indices of the data matrix were partitioned between processors.

The generally constant value of the product of the number of processors and the run time indicates that the scaleability is near-perfect under these conditions, except with only three processors, where the poor result obtained is probably due to the necessarily poorly-matched topology, in which there has to be a one-dimensional partition with one portion of the task requiring twice as much communications traffic as the other two. The comparative results with different topologies for eight processors are interesting, as the differences are small, although the (nominally) optimum topology ($2\times2\times2$) actually gives the poorest result, albeit by a negligibly small margin. It is more interesting to note that the least optimal partition ($1\times8\times1$) shows no degradation of efficiency in this case, indicating that processor workload imbalance is negligible in this case, and that the 'slice' partitions are not so thin as to be dominated by communications. These particular results may have been influenced by the relatively 'long and thin' problem space considered. Amdahl's law suggests that the scaleability will be degraded with a large number of processors, and this effect is starting to appear with sixteen processors, but not to a great degree.

Table 3. Scaleability tests with Meiko CS-2 computers

p	f_c. MHz	Run time mins	Top-ology	p×time (mins)	Speed-up†	Efficiency †
3	50	65:33	1×3×1	196.7	1.5	49%
4	50	24:00	2×2×1	96	4	100%
6	50	16:57	2×3×1	101.7	5.7	94%
8	50	12:48	2×2×2	102.4	7.5	94%
8	50	12:40	2×4×1	101.3	7.6	95%
8	50	12:41	1×8×1	101.5	7.6	95%
16	40	8:12	2×4×2	105.0*	14.6*	91%

†Assumes ideal scaleability for ≤4 processors. *Clock frequency (f_c) normalised to 50 MHz.

The head-telephone interaction model was ultimately run using the parallel FDTD code on a 128-processor IBM SP-2 parallel computer to assess the effectiveness of the parallel port. Figure 4 shows three sets of results for speed-up of the code. Test 1 was for a 2.5 mm resolution model of the head, hand and mobile telephone, Tests 2 and 3 being at resolutions of 2 mm and 1 mm respectively. Table 4 shows the FDTD problem size for each of these test cases and Fig. 5 shows a typical slice of the computed output.

It should be noted that it was not possible to run the 1 mm problem on fewer than eight processors on this computer since the data quantity then became too large for the memory of the processors in use. Therefore the results for one, two and four processors were extrapolated from the result at eight processors using the efficiencies found in the 2 mm test case. The curve labelled 'ideal' in Fig. 4 illustrates the speed-up that would be achieved with a 100% efficient system.

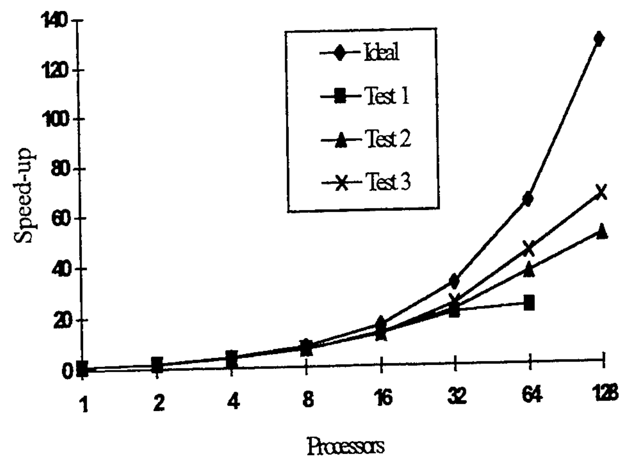


Figure 4: Speed-up results from the head-telephone test cases, running on a 128-processor IBM SP-2.

Table 4. Memory Requirements for Test Problems

Test No.	Resolution	Problem Size (cells)	Memory Required
1	2.5 mm	1.3×10^6	45 MBytes
2	2 mm	2.5×10^6	90 MBytes
3	1 mm	20×10^6	720 MBytes

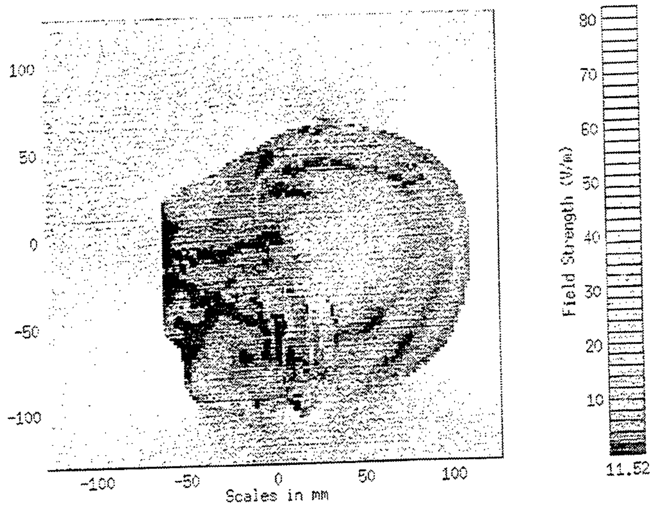


Figure 5 A typical slice of the computed output: electric field magnitude through the centre of a head adjacent to a 1.8GHz mobile telephone handset.

Figure 4 demonstrates that efficient massively-parallel processing of electromagnetic problems is now a mature and viable technique, whereas early efforts frequently showed little gain after a very small number of processors was exceeded. It also shows two very important effects: firstly, the larger the computational task, the greater is the efficiency of the parallelisation, because the ratio of computational to communications tasks is increased. Secondly, when a task is subdivided over too many processors the efficiency of the parallelisation reaches a limit because the computational task is attenuated to a point where it becomes of the same order as the communications task: this is clearly seen in the results for the smallest test case (Test 1) with 64 processors.

The objective of portability has been demonstrated by use of this software on Meiko and IBM computers, but it was also successfully tested on one by Parsytec, and on networks of UNIX workstations.

5 CONCLUSIONS

Electromagnetic field computation is inherently a SPMD (single instruction, multiple data) process which is very appropriate for execution on vector and parallel supercomputers. Automatic algorithms for efficient exploitation of these are not yet fully developed and some care is thus necessary in the parallelisation of traditional serial software, taking account of the structure of the software and the way in which it interacts with the detailed architecture of the computer. Ways of doing this have been reviewed: for differential-equation based methods the PVM or MPI algorithms appear to offer a useful and portable approach which is accepted over a wide range of platforms.

Provided a sufficiently large task is addressed, it has been shown that efficient massively-parallel processing of electromagnetic problems is now a viable and mature technique, at least for differential-equation based formulations, whereas early efforts frequently showed little gain after a very small number of processors was exceeded.

It was found that as the problem size increased the efficiency of the code increased, the results tending towards the ideal. It is a paradox that it is difficult to accurately assess the efficiency of very large test cases, as they are too large to be run on a small number of processors. It was observed that when a task is subdivided over too many processors the efficiency of the parallelisation reaches a limit because the computational task is reduced to a point where it becomes of the same order as the communications task.

Acknowledgements

Primary financial support was provided by the UK Engineering and Physical Sciences Research Council (EPSRC). The human head dataset was created by Peter Olley, of Bradford University, using MRI data from the University of North Carolina. Access to the KSR-1 computer was provided by the Manchester Computer Centre. The parallelisation of EMA3D was funded by the ESPRIT III EUROPORT2 project and undertaken jointly with British Aerospace (Military Aircraft) Ltd and EMA Inc. Access to IBM and Meiko parallel computers was arranged by Smith System Engineering Ltd and provided by CERFACS (Toulouse), CERN (Geneva), Cornell Theory Center (Ithaca) and KTH (Stockholm).

References

[1] Olley, P., and Excell, P. S., 1995: 'Classification of a High-Resolution Voxel Image of a Human Head', ***International Workshop on Voxel Phantom Development***, National Radiological Protection Board, Chilton UK, 16-23.

[2] Davidson, D.B. and Ziolkowski, R.W., 1995: 'A Connection Machine (CM-2) Implementation of a Three-Dimensional Parallel Finite Difference Time-Domain Code for Electromagnetic Field Simulation', ***Int. J. Num. Modelling: Electronic Networks, Devices and Fields***, **8**, 221-232.

[3] Rodohan, D.P., Saunders, S.R., and Glover, R.J., 1995: 'A Distributed Implementation of the Finite Difference Time-Domain (FDTD) Method', ***Int. J. Num. Modelling: Electronic Networks, Devices and Fields***, **8**, 283-291.

[4] Buchanan, W.J., Gupta, N.K., and Arnold, J.M., 1993: 'Simulation of Radiation from a Microstrip Antenna using Three-Dimensional Finite-Difference Time-Domain (FDTD) Method', ***Proc. IEE Int. Conf. on Antennas and Propagation, Edinburgh***, **2**, 639-642.

[5] Amdahl, G.M., 1967: 'Validity of Single-Processor Approach to Achieving Large-Scale Computing Capability', ***Proc. AFIPS Conf., Reston, VA, USA***.

[6] Tatalias, K.D., and Bornholdt, J.M., 1989: 'Mapping Electromagnetic Field Computations to Parallel Processors', ***IEEE Trans. Magnetics***, **25**, 2901-2906.

[7] Beguelin, A., et al, 1991: 'A Users' Guide to PVM - Parallel Virtual Machine', ***Oak Ridge National Laboratory***, USA, Report No. ORNL/TM-11826.

[8] Heath, M.T., and Etheridge, J.A., 1991: 'Visualizing the Performance of Parallel Programs', ***IEEE Software***, **8**, 29-39.

[9] Excell, P.S., Porter, G.J., Tang, Y.K., and Yip, K.W., 1995: 'Re-working of Two Standard Moment-Method Codes for Execution on Parallel Processors', ***Int. J. Numerical Modelling: Electronic Networks, Devices and Fields***, **8**, 243-248.

[10].Gedney, S.D., and Barnard, S., 1995: 'Efficient FD-TD Algorithms for Vector and Multiprocessor Computers', *in* ***Taflove, A., 'Computational Electrodynamics: The Finite-Difference Time-Domain Method'***, Boston: Artech House.

[11] Holland, R., 1977: 'THREDE: A Free-Field EMP Coupling and Scattering Code', ***IEEE Trans. Nuclear Science***, **NS-24**, 2416-2421.

[12] Gedney, S.D., 1995: 'Finite-Difference Time-Domain Analysis of Microwave Circuit Devices on High Performance Vector/Parallel Computers', ***IEEE Trans. Microwave Theory & Tech.***, **43**, 2510-2514.

[13] Varadarajan, V., and Mittra, R., 1994: 'Finite-Difference Time-Domain (FDTD) Analysis using Distributed Computing', ***IEEE Microwave and Guided Wave Lett.***, **4**, 144-145.

[14] Perala, R., Whittle, S., Hargreaves, M., and Kerton, P., 1995: 'An Introduction to PEPSE ('Parallel Electromagnetic Problem Solving Environment') and Considerations for Paralellization of a Finite Difference Time-Domain Solver', ***Int. J. Num. Modelling: Electronic Networks, Devices and Fields***, **8**, 187-204.

[15] Excell, P. S., Olley, P., and Jackson, N. N., 1996: 'Modelling of an Arbitrarily-Oriented Mobile Telephone Handset in the Finite-Difference Time-Domain Field Computation Method', ***Applied Computational Electromagnetics Society Journal***, **11**, 55-65.

A Dedicated TLM Array Processor

D. Stothard, S.C. Pomeroy
Dept. of Electronic and Electrical Engineering, Loughborough University, Loughborough
Leicestershire, U.K, LE11 3TU

ABSTRACT. The transmission line matrix (TLM) method is introduced and the specific issue of computational efficiency is discussed. The implementation of TLM on parallel computers is studied leading to the creation of a highly efficient processor designed specifically for TLM. Limitations introduced by the connection strategies employed by most parallel architectures are overcome through the use of a novel data routing architecture. The basic idea is extended to include stub loaded and three-dimensional TLM. The development of a prototype processor is discussed and potential applications are given.

1 Introduction

Analytical solutions to Maxwell's equations are possible only in restricted cases. The advent of the digital computer has given rise to a number of numerical techniques for solving Maxwell's equations and modelling electromagnetic wave phenomena. Of these the most common are the finite difference time domain (FD-TD), finite element (FE), and transmission line matrix (TLM) methods[1]. Finite difference and finite element methods respectively perform differentiation or integration of the electric or magnetic field over a defined region. In contrast, TLM is built around an array of connected secondary radiators, giving discretisation in both space and time. Thus TLM bears a uniquely close physical resemblance to the processes of propagation. TLM in fact offers a solution to the Telegraphers equation and has been applied to both propagation and diffusion modelling.

This paper is divided in to eight sections. After this introduction, Section 2 gives a brief introduction to the TLM method and demonstrates the key factors leading to lengthy run times. Section 3 looks at the implementation of TLM through parallel and distributed computing methods and analyses the reasons for the worse than expected performance increases shown by such methods. Section 4 introduces a new scatter processor designed specifically for the two dimensional TLM method and discusses the implications of working with the new design. A new strategy for mapping a TLM array in to hardware is introduced in section 5. The processes developed in sections 4 and 5 are extended to cover stub loaded and three dimensional TLM arrays in section 6. Section 7 details a general processor utilising reconfigurable logic to optimise performance for each of the above TLM schemes within a single architecture. The design and testing of a prototype processor is discussed in section 8. The conclusions in section 9 look towards the future of the processor and suggest potential applications.

2 Review of TLM

A basic introduction to the workings of TLM is given here. Fuller, more rigorous derivations, and discussion of the relationship between TLM and other techniques are contained in references [1,2,3,4,5,6,7]

The transmission line matrix (TLM) method, first reported by Johns and Beurle[4] in the early 1970s, offers a simple and unconditionally stable method for realising time domain solutions to propagation and diffusion problems. The basic building block for the two dimensional (2D) TLM method for waves is the shunt node [5], formed by an orthogonal junction between two ideal transmission lines of length Δl, (Figure 1)

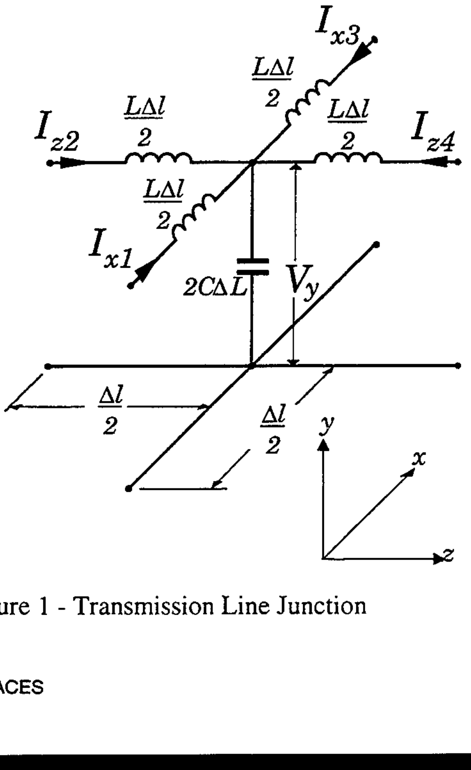


Figure 1 - Transmission Line Junction

This unit cell is repeated to fill the region under consideration with a Cartesian mesh of transmission lines.

An impulse travelling towards a node in the mesh will see an impedance mismatch at the junction and will be scattered according to (1).

$$\begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \end{bmatrix}^r = \frac{1}{2}\begin{bmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \end{bmatrix}^i \qquad (1)$$

where $V_{1\text{-}4}$ are the voltage impulses in the branches 1-4 and the suffixes *i* and *r* denote incident and reflected impulses respectively. (1) is more commonly expressed as

$$V_n^r = \frac{1}{2}\sum_{m=1}^{4} V_m^i - V_n^i \qquad (2)$$

It can be shown[5] that there exists a direct relationship between the voltages and currents on the transmission lines and the electric and magnetic fields in the region modelled by the mesh. Impulses traverse the transmission lines with a fixed velocity, *u*, thus all impulses scattered from a node will become incident upon the neighbouring nodes after a time

$$\Delta t = \frac{\Delta l}{u}$$

The implementation of TLM in software follows an iterative process,

i) impulses are injected in to the mesh by exciting the appropriate voltages or currents.

ii) Equation (1) is applied to the incident data at each node using an instruction loop.

iii) A second loop passes the data scattered from each node to the neighbouring nodes for which it forms the incident impulses in the next iteration.

The addition of another node to the mesh requires one further application of the scatter and connect loops, thus processing time increases linearly with the model size.

It has been demonstrated [5] that the propagation velocity of a wave through the mesh is dependent upon the direction of travel and the mesh discretisation, Δl. Propagation at 45° to the axes is unperturbed, however axial propagation is frequency dependent, giving rise to dispersion. The plot of the dispersion characteristic shows that at least 10 nodes are required per wavelength at the highest frequency under consideration to reduce dispersion in the mesh to an acceptable level. This restriction upon Δl, along with the need for fine meshes to accurately model detailed geometries, can lead to very large meshes with many nodes. This produces a correspondingly large run time for TLM code on serial computers.

The basic scatter process of (1) may be adapted to model propagation in inhomogenous and lossy media through the addition respectively of capacitive or absorptive stubs of length $\Delta l/2$ to the node. Some of the energy at the node is scattered in to the stub and then, in the case of capacitive stubs, returned to the node in the next iteration. This changes the form of the scattering equation to

$$V_n^r = \left\{\frac{2}{y}\left[\sum_{m=1}^{4} V_m^i + y_0 V_s^i\right]\right\} - V_n^i \qquad (3)$$

Where $y = 4 + y_0 + g_0$, y_0 and g_0 are the normalised capacitive and lossy stub impedances respectively. Because the stub energy is not passed to neighbouring nodes the connection process remains unchanged.

The most common scheme for three dimensional (3D) modelling is the symmetrical condensed node (SCN) [6] shown in fig. 2. The scattering matrix of the SCN is a sparse 12 x 12 matrix which has discrete solutions of a form similar to (2). As with 2D modelling, scattered data are passed to the neighbouring nodes.

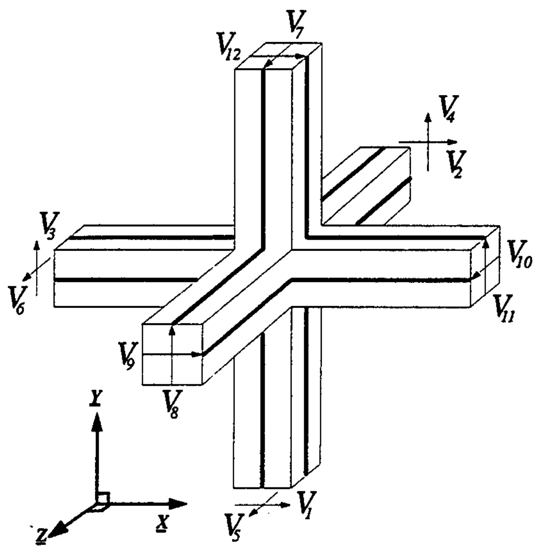


Figure 2 - The Symmetrical Condensed Node

3D modelling of inhomogenous media is possible through the addition of stubs to the SCN as with the 2D shunt node. However, recent developments have produced more computationally efficient 3D node schemes[7] such as the symmetrical super condensed node (SSCN). These schemes remove the need for some or all of the stubs, reducing the data storage requirements for each node and producing more efficient scattering algorithms with simple, discrete solutions.

The introduction of more complex arrays for inhomogenous media and 3D meshes add further to the run times for TLM. 3D modelling generally requires considerably larger arrays with more nodes than 2D modelling. The addition of stubs to the array complicates the scattering process and increases the number of operations required to perform each scattering operation, thus increasing the time taken for each loop of the scatter process.

3 Review of Parallel Implementations

The scatter and connect processes in TLM are explicit operations. To perform the operation at each node requires only data from that node. This means that the processes may be applied simultaneously to each node without conflict. This inherent parallelism has been exploited in the past by implementing TLM through parallel computing[8,9,10,11,12,13]. A variety of processing elements and connection strategies have been used. However a number of features are common to most or all of the implementations.

i) There is a direct physical mapping of the mesh to the parallel processor, i.e. each processing element maps to one node in the mesh. Interconnections are generally simple near neighbour links.
ii) The simple nature of the TLM algorithm leaves much of the processing power of the parallel processors unused.
iii) Limited bandwidths and TLM's low ratio of computation to I/O cause communication bottlenecks, reducing performance.
iv) The hardware requirements of each of these methods place them beyond the reach of most researchers.

Whilst it is clear from the literature that implementing TLM on a parallel computer can produce significant performance increases it is also clear that modification of the nature of the implementation could produce further performance improvements. There are 3 key points that must be addressed.

i) The direct mapping of one node in the mesh to one processing element limits the size of the model to the number of processing elements available.
ii) The granularity of the processing elements must match that of the problem for efficient performance.
iii) The bandwidth provided must be sufficient to prevent bottlenecks during the connection phase.

The best way to ensure a granularity match is to use a processing element designed specifically for TLM. The use of application specific processors to provide efficient computing performance has grown rapidly in recent years. There have been several application specific TLM processors designed in the past, these can be placed in to two categories.

i) Single node coprocessors.
ii) Complete arrays.

The former approach, developed by Saleh[14], uses a single node which is utilised as a coprocessor by the software. Each time the software encounters a scattering operation the incident data is passed to the coprocessor and the scattered data is returned to the host. Scattering remains a serial process, each node is treated individually, and the performance increase comes from the efficiency of the reduced instruction set (RISC) architecture of the node processor. This approach has the advantage that the array size is limited only by memory availability.

The latter approach, as used by Gregory[15], provides an array of RISC processors on to which the TLM mesh is mapped. The host system provides initial data and reads out results from the array. Scatter and connect are performed in parallel providing a significant performance increase, however the main advantage of the system is in the efficiency with which each scatter computation is performed. Each processing element has been designed to perform only the TLM algorithm and is therefore fully utilised the whole time. The application specific approach, while more efficient, has the disadvantage that the processor may be limited to one type of TLM calculation, e.g. stub loaded 2D.

4 Design of a Scattering Processor

When developing a new application specific processor for TLM it is possible to consider the scatter and connect processes separately. The choice of architecture, i.e. coprocessor or complete array is important. It appears that the array architecture provides higher throughput as it performs some operations in parallel, however this is offset by the higher bandwidth and hardware requirements. The design of the processing elements which perform the scattering operations will go a long way towards deciding which is the most efficient architecture.

The design process begins with the development of a suitable algorithm for hardware implementation. Numerical devices such as multipliers are difficult to implement in logic and can lead to large, slow circuits where as adders and subtractors are easily constructed. Nodal schemes such as the shunt node and the SCN require only additions, subtractions and a divide by two, which may be implemented by shifting in binary, therefore for these schemes the most suitable algorithm is the one which minimises the number of addition and subtraction operations. The modelling of variable media using either stub loading or the SSCN requires multiplication. Because the multiplier is the dominant component, minimisation of the number of multiply operations becomes the overriding concern.

The optimised algorithm must be developed in to a suitable hardware configuration. This process is simplified through the use of behavioural modelling and the VHDL hardware description language. This allows a circuit to be described in terms of its behaviour (in this case the chosen TLM algorithm) and its performance simulated. Tools are then available to synthesise a gate level circuit description from the behavioural VHDL. This circuit may again be tested through simulation before production takes place. A logical starting point is the design of a simple two dimensional TLM system.

The design of the scattering processor is a trade off between a number of conflicting requirements. Should the processor have a RISC architecture or should the scatter process be mapped directly to the hardware? The former provides more flexibility but the latter will be faster. Flexibility is a key issue; although most of the mesh is homogenous, boundaries, sources and targets all require handling differently. Specialised nodes for boundaries etc. are one solution, however these would either be placed at fixed locations within the mesh or would require a complex routing procedure to allow arbitrary placement. A more viable solution is a generalised processor which can act as a simple scattering point, a source, target or boundary node as required.

An early attempt by the authors to design a TLM scatter processor is documented in [16]. This simple design was a direct mapping of the 2D scatter equation on to a field programmable gate array (FPGA). Despite producing very high throughput the design failed in a number of key areas.

i) The processor would only perform a simple scattering operation. Boundaries etc. were untreated.
ii) The data parallel design requires a very high bandwidth.
iii) There is no access to data within the array of processors, only data reaching the edge of the array can be read. Similarly there is no access to the total incident energy data commonly used to visualise propagation within the mesh.
iv) The word length used is fixed by the width of the logic.
v)

In order to overcome these problems a new design has evolved which utilises a pipelined, bit serial architecture. A block diagram of the processor is shown in fig. 3.

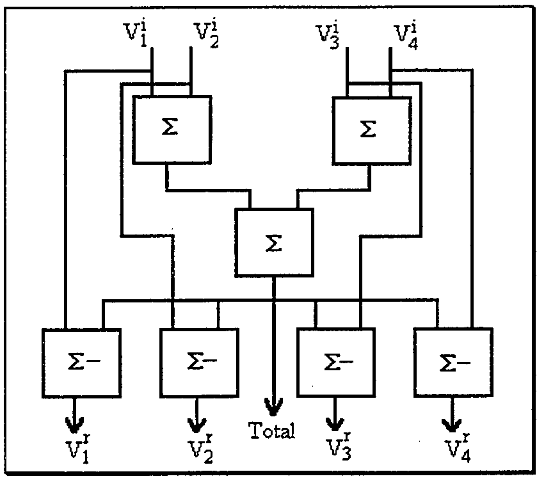


Figure 3 - Block Diagram of the TLM Processing Element

Σ indicates a single bit wide full adder, Σ- indicates a single bit wide subtractor. The operation of the processor is simple. The first two levels add together the input data and the result is divided by two by discarding the first bit output from level 2. The sum is then routed to four subtractors which produce the output values. A single pin output provides access to the total incident energy data. By varying the timing of the control signals this processor is capable of operating on data of any word length. The bit serial design also reduces the required bandwidth considerably. The design has one further advantage in that it is able to perform simple boundaries (those with reflection coefficients of $\rho = 0, 1, -1$). This is done by incorporating the boundary in to the scatter equation thus

$$V^r_m = \rho\left\{\frac{1}{2}\sum_{n=1}^{4} V^i_n - V^i_m\right\} \qquad (4)$$

The data is preceded by a two bit code which is used to define what type of boundary is present. The code is added to all data words allowing the arbitrary placement of boundaries within the TLM mesh.

5 Mapping the Connect Process

The connect process is required to provide near neighbour connection between the nodes in the TLM mesh. Previous systems have either performed this mapping in software or have produced a large physical array with hardwired interconnects between the processors on to which the TLM mesh is mapped. The former approach is slow whereas the latter requires a large number of processors and limits the size of the mesh which may be implemented. A more flexible approach would be to have an architecture which allows a mesh of arbitrary size to be mapped on to a fixed number of processors.

The localised nature of the connect process means that any given node can communicate only with the adjacent nodes in its own row of the mesh or with the adjacent nodes in the rows immediately above and below it. Thus only three rows of data are active at any given time. Scattered data will either be passed to a neighbouring node or, if a boundary is present, it will be returned to the node from which it was scattered. This information has been utilised to develop a hardware mapped connect process. Three small blocks of memory, called the active lines, are used, holding copies of data from three adjacent rows in the array. Data is scattered from the centre row of the three and the output from the processors is sent to a logic cell which reads the boundary flag attached to each data word and routes the data to either the correct adjacent node or back to the scattering node as appropriate. If the number of processors available are less than the number of nodes in each row of the model then another memory block is required to hold data scattered left and right from the edges of the row of processors until it is required.

From the scatter and connect processes described above it is simple to produce a complete system [17] for the solution of a two dimensional TLM mesh. A large main memory is required to store all the current data in the array, that is the four incident values at each node in the current iteration. Starting at one corner of the mesh data from the first N nodes (where N is the number of processors available) are passed to the scatter processors. These output the total energy incident upon the nodes, which may be read out to a host system or stored in a further memory, and the scattered data values which are passed to the connection logic and associated memory. At the end of each row processing moves on to the start of the next row and continues. Data held in the third row of the connection memory is no longer affected by the current scattering events and can be written back to the main store where it forms the incident data for the next iteration. A block diagram of the system is shown in fig. 4.

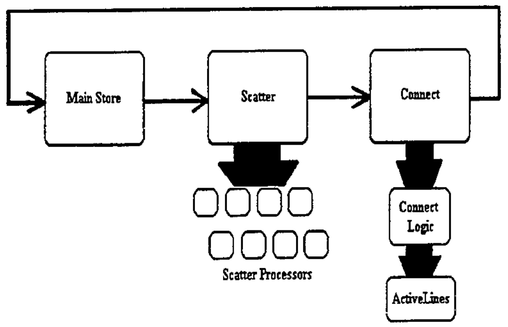


Figure 4 - Hierarchic Block Diagram of the TLM System

This unique mapping of the connect process allows a small number of processors to be mapped to a mesh of arbitrary size. This approach has the advantage of performing the scatter and connect processes partially in parallel, thereby increasing performance, but the mapping of a large mesh on to a small array of processors eliminates the restrictions on model size imposed on most parallel TLM applications.

6 Extending System Capabilities

The processor described above is capable of performing only basic 2D TLM on a homogeneous mesh, however the techniques it introduces may be developed to produce similar systems capable of more complex operations. An extension of the basic method to allow for stub loaded 2D modelling is straightforward and only requires modification of the scatter processors. The stub loaded scatter processor is similar to that of the basic processor except for the addition of a multiplier which is preloaded with the value of y_0 for the next node while processing takes place on the current data. In addition to this modification a small amount of memory is required to hold both the energy in the stub at each node and the node parameters y and y_0. This data is not connected to neighbouring nodes, therefore the connect process remains unchanged.

The techniques used in two dimensional modelling may be extended to three dimensional modelling. The connect process in three dimensions can be seen as identical to the two dimensional connect process with additional data passed to nodes in the planes above and below the scattering node. This can be accomplished using the memory architecture shown in fig. 5.

The active lines and main store are identical to the active lines and main store in the 2D system. Data from the active lines is used to update the main store along with the data scattered in to the last plane from the scattering processors.

The scattering processor requires some modification. Three dimensional nodes are 12 port devices thus the processors require 12 input data words as opposed to 4 for the 2D node. The architecture of an SCN scatter processor is identical in its operation to the basic 2D scatter processor using pipelined layers of adders and subtractors to produce the output data. A further output

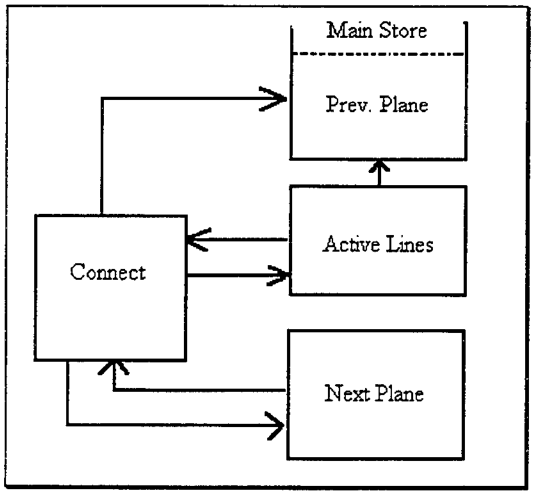


Figure 5 - 3D Connection Memory Architecture

provides access to one of the field components, selected before processing begins, for visualisation purposes. The SSCN, used for isotropic, inhomogeneous three dimensional media, requires the storage of two impedance values for each node. These can be treated in a similar way to the stub data for the 2D stub loaded processor, being held in a separate memory. As with the 2D stub loaded processor, preloading of the multipliers with the impedance values improves performance.

7 A General, Reconfigurable Processor

Consider the architecture of the processor required to perform 3D TLM using the SSCN, fig. 6(a). By utilising only the required components from this architecture we arrive at the processor architectures for the three other classes of TLM processor, basic (SCN) 3D, stub loaded 2D and basic 2D, fig. 6(b-d). This illustrates not only the close physical similarity between all TLM schemes, but also the fact that one architecture, with a little modification, should be capable of performing any of the four key TLM schemes. There are two ways of allowing the necessary modifications to the scatter and connect hardware, reprogrammability or reconfigurability. A reprogrammable processor decodes instructions and performs the operations dictated by its program. This approach is slower than the hardware mapped systems developed above as it requires instruction fetching and decoding cycles. This approach would also, naturally, introduce some redundancy as not all features are implemented in each scheme. A more suitable option is to use reconfigurable logic such as a field programmable gate array (FPGA)[18], which is configured at start up but may be given a different configuration each time it is used. The authors use Xilinx FPGAs as the granularity of the configurable logic blocks (CLBs) is well suited to the problem, reducing redundancy in the design. The system requires three reconfigurable components, scatter, connect and control. Using the architecture of figure 6(a) the most suitable processor configuration can be chosen for each particular problem, optimising performance. The operation of each processor is identical as far as the user is concerned, each processing scheme requires the same set of control signals. The configuration for each FPGA can be stored on EPROM or as a file on a host system, therefore it is possible to build up a library of common configurations which may be combined as necessary depending upon the given problem.

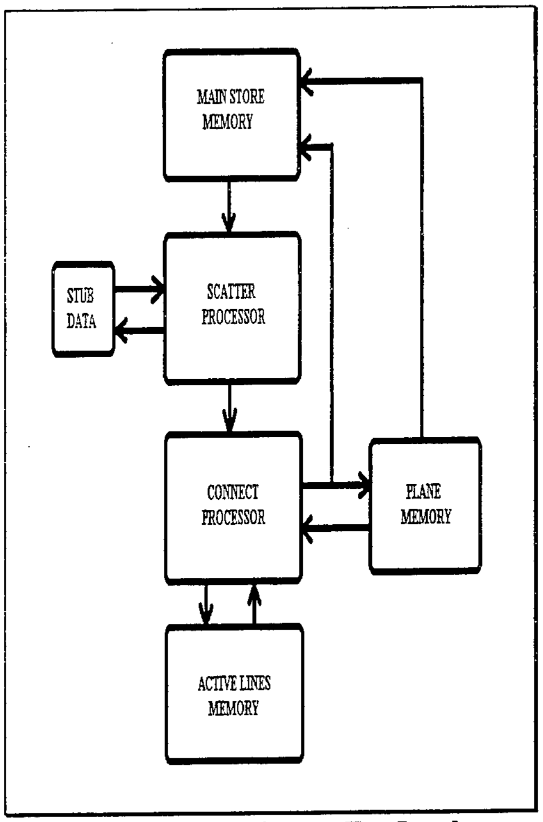


Figure 6a - SSCN System Top Level

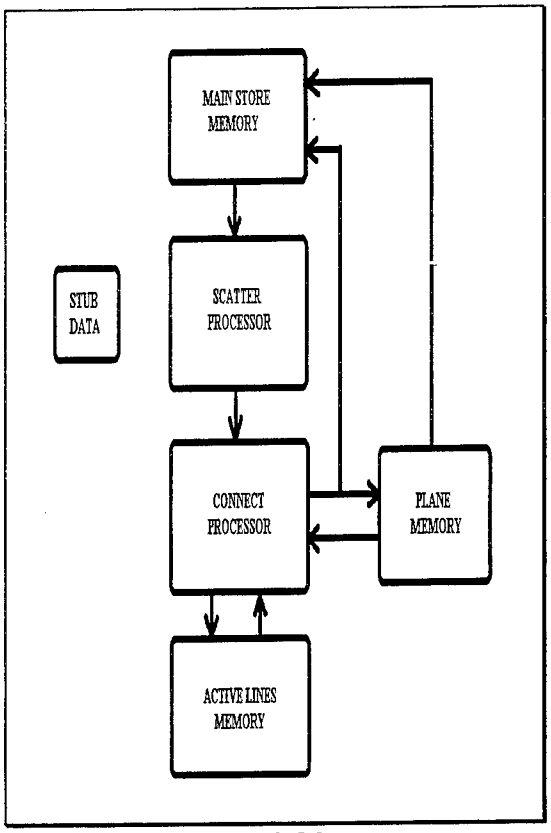


Figure 6b - SCN System

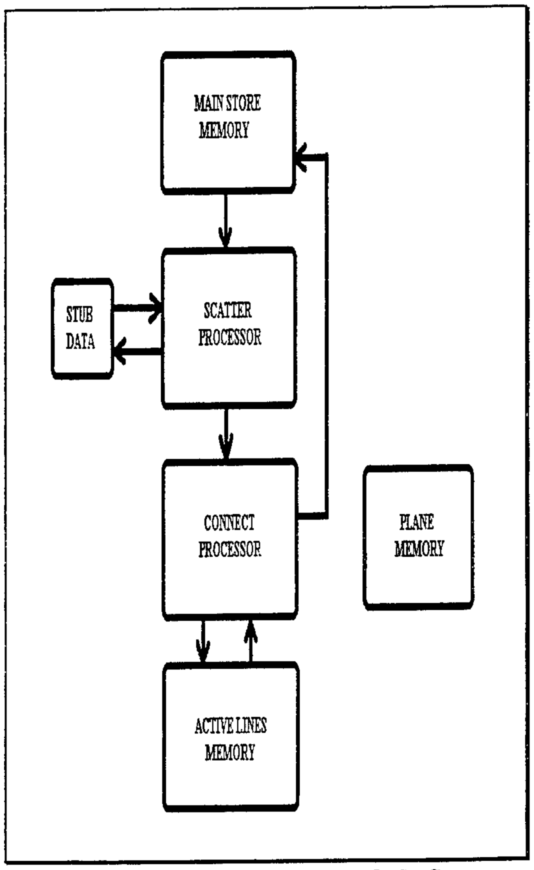


Figure 6c - Stub Loaded Shunt Node System

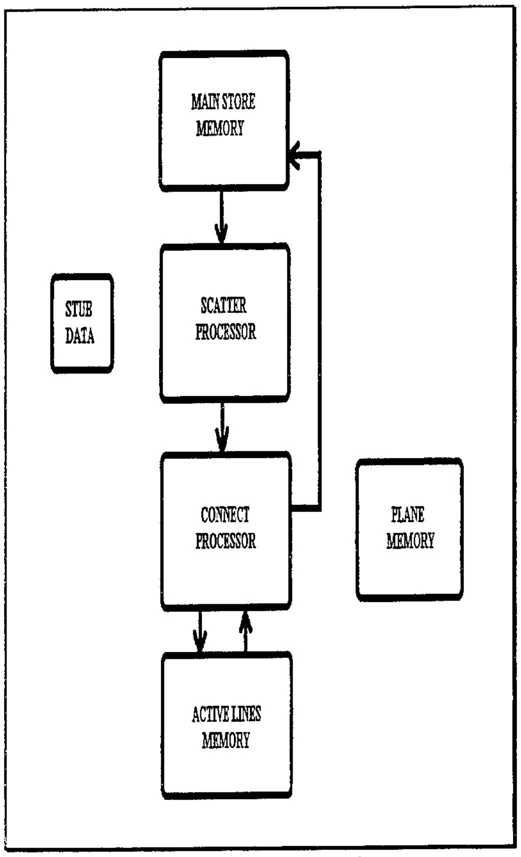


Figure 6d - Shunt Node System

8 Testing and Results

Modelling of the system using the VHDL hardware description language has allowed for extensive testing. Synthesis of several processor configurations realised using both behavioural and structural synthesis has allowed optimisation of each component in the system. The scattering processors have been tested on a Xilinx XC4013 and have demonstrated correct operation. A prototype system limited to performing basic 2D TLM is under construction. This will allow more rigorous testing of the system to be carried out and will provide useful performance measures. The prototype has taken the form of a PCI card which may be hosted on a personal computer. Initialisation data is fed in to the system which then runs in the background, allowing the host to perform other tasks. Output data may be read from the system either after every iteration or at the end of the processing run and access is provided to data from individual nodes in the model with little communication overhead. Predictions of current operating rates suggest that a scattering component with 8 processors will be capable of performing up to 8 million scattering events per second on 32 bit data. This compares favourably with $6x10^5$ scattering events per second achieved using serial code on a 200Mhz Pentium Pro equipped PC. This may be improved by partitioning the model between two or more boards and allowing them to operate concurrently. As with software implementations there is a linear relationship between run time and model size. This relationship is inversely proportional to the number of scattering processors available, hence increasing the number of scatter processors can have a pronounced effect on throughput. While the current prototype contains only 8 scatter processors a working system may contain several thousand.

9 Conclusions

The system has two potential areas of application, those where large arrays must be processed and those where smaller arrays must be processed quickly. Possible applications in the first group include EMC studies and ultrasonics/acoustics. The second type of application includes time critical operations such as medical imaging, real time remote sensing etc.

The unique nature of the connect process described above allows the system to operate on any algorithm which requires a simple near neighbour data transfer. The scatter processor is replaced with a processor capable of performing the new algorithm. These include diffusion modelling using a link line TLM scheme and non TLM applications such as FD-TD and cellular automata. Through the use of hardware description languages and logic synthesis the development of scatter processors for these other techniques is relatively simple. Indeed in some cases existing software routines may be transferred in to VHDL and thus directly in to hardware.

10 Summary

A complete application specific processor for the solution of the transmission line matrix (TLM) method has been presented which offers a considerable reduction in run times while overcoming the limitations imposed through conventional parallel architectures. A unique mapping of the TLM connect process to hardware allows a small number of scattering processors to process a mesh of any size. Through the use of reconfigurable logic the processor may be optimised to perform any of the four main TLM schemes, 2D basic, 2D stub loaded, 3D SCN, 3D SSCN without requiring any user reprogramming.

It is interesting to note that a processor developed entirely from the TLM equations may, through the use of reconfigurable computing, be applied to many other numerical modelling techniques.

A prototype system is under construction which is expected to provide throughput an order of magnitude greater than current software implementations while future, large scale systems may offer performance far beyond this point. Realisation of the system as a PCI card for a personal computer makes it an accessible, low cost alternative to traditional parallel computing.

References

[1] 'Numerical Techniques for Microwave and Millimeter-Wave Passive Structures' T.Itoh - Ed. J.Wiley, 1989

[2] Russer, "On the Field Theoretical Foundation of the Transmission Line Matrix Method", 1st International TLM Workshop, University of Victoria, Canada, 1-3 Aug. 1995

[3] Simons and Lovetri, "Derivations of Two-Dimensional Algorithms on Arbitrary Grids Using Finite Element Concepts", 1st

International TLM Workshop, University of Victoria, Canada, 1-3 Aug. 1995

[4]P.B. Johns & R.L. Beurle, 'Numerical Solution of 2-Dimensional Scattering Problems using a Transmission-Line matrix' Proc. IEE, Vol.118(9), pp.1203-08, 1971

[5] W.J.R. Hoefer 'The Transmission line Matrix (TLM) Method' in 'Numerical Techniques for Microwave and Millimeter-Wave Passive Structures' Chapter 8, pp.496-591, J.Wiley, 1989

[6] P.B. Johns 'New Symmetrical Condensed Node for Three Dimensional Solution of Electromagnetic wave Problems by TLM' Electronics Letters, Vol.22(3), pp.162-164, 1986

[7] V. Trenkic 'The Development and Characterisation of Advanced Nodes for the TLM Method' PhD Thesis, University of Nottingham, 1995

[8] P.P.M So; C. Eswarappa & W.J.R Hoefer 'Parallel and Distributed TLM Computation with Signal Processing for Electromagnetic Field Modelling' Int. Jnl. Num. Mod : Elect. Networks, Devs. And Fields, Vol.8, pp.169-185, 1995

[9] J.L Dubard; O. Benevello; D. Pompei; J. Le Roux; P.P.M So & W.J.R Hoefer 'Acceleration of TLM Through Signal Processing and Parallel Computing' Int. Conference on Computation in Electromagnetics, IEE, pp.71-74, 1991

[10]P.O Luthi; B. Chopard & J-F Wagen 'Wave Propagation in Urban Microcells : a Massively Parallel Approach Using the TLM Method' Applied Parallel Computing in Physics, 2nd International Workshop, pp.408-18, 1995

[11] C.C Tan & V.F Fusco 'TLM Modelling Using an SIMD Computer' Int. Jnl. Num. Mod : Elect. Networks, Devs. And Fields, Vol.6, pp.299-304, 1993

[12] P.J Parsons; S.R Jaques; S.H Pulko & F.A Rabhi 'TLM Modelling Using Distributed Computing' IEEE Microwave and Guided Wave Letters, Vol.6(3), pp.141-42, 1996

[13] A. Mallik 'Parallel Executing TLM Code and it's Application to EMC and the Motor Vehicle' IEE Colloquium on EMC and the Motor Vehicle, pp.3/1-4, 1990

[14] A.H. Saleh 'A Dedicated Processor for Solving TLM Field Problems' PhD Thesis, University of Nottingham, 1982

[15] Gregory, S 'Design of a Single Bit Processor for TLM Using Full Custom IC Design' Dissertation (BEng), University of Nottingham, 1989

[16] D. Stothard & S.C. Pomeroy 'An Application Specific Processor for TLM' 1st International TLM Workshop, pp.277-280, University of Victoria, Canada, 1-3 Aug. 1995

[17] D. Stothard & S.C. Pomeroy 'An Application Specific TLM Array Processor' TLM - the Wider Applications, pp.3.1-3.5, University of East Anglia, Norwich, 27 June, 1996

[18] 'The Programmable Logic Data Book' Xilinx Inc, 1994

Bandwidth Reduced Full-Wave Simulation of Lossless and Thin Planar Microstrip Circuits

A. Caproni, F. Cervelli, M. Mongiardo, L. Tarricone, F. Malucelli

Istituto di Elettronica, Via G. Duranti,93, 06131, Perugia, Italy.
Corresponding Author: L. Tarricone, tarricone@istel.ing.unipg.it

Abstract— We present a full-wave, high-performance, numerical scheme for the analysis of planar microstrip circuits which is based on an efficient electromagnetic formulation of the field problem and on the bandwidth reduction of the discretized sparse matrix.

The above mentioned electromagnetic efficiency is attained by considering a Mixed Potential Integral Equation (MPIE) with the kernel expressed by closed-form spatial-domain Green's functions; as a consequence, the reaction integrals are evaluated by using just one-dimensional numerical integration over a finite spatial domain. Moment method discretization of the MPIE leads to the corresponding matrix problem.

The accurate analysis of the matrix properties shows that a sparsity of 70-85 % in the discretized linear system can be routinely enforced without significantly altering the solution accuracy.

A new scheme for the sparse matrix bandwidth reduction, particularly tailored for electromagnetic problems, can be therefore introduced, leading to considerable reductions of the simulation time. Results are presented demonstrating that the use of a bandwidth reduction strategy coupled with efficient problem-matched Green's functions allows as to obtain speed-ups in simulation time of more than one order of magnitude with respect to standard state-of-the-art implementations.

I. Introduction

The efficient and rigorous analysis of microstrip circuits, including patch antennas and printed dipoles, requires the use of appropriate Green's function representation, adequate choice of basis functions for field expressions, and, last but not least, efficient strategies for the solution of the linear system resulting from the integral equation discretization.

Recently, by means of an ingenious device, a novel approach has been developed [1] for obtaining closed-form expressions for the spatial domain Green's functions corresponding to the vector and scalar potentials associated with a horizontal electric dipole located over a tick substrate. The technique has been furtherly extended in [2], [3] and leads to substantial savings of computation time when analizing planar microstrip configurations by variational techniques, such as the method of Moments (MoM). In this work we have therefore used the latter Green's functions, hence significantly reducing the amount of time necessary for the impedance matrix filling.

Concerning the choice of suitable basis functions, although it has been shown in several occasions that the inclusion of the appropriate edge singularity significantly enhances convergence properties [4], in order to keep the geometries as flexible as possible, standard roof-top expansions have been used.

However, the suitable selection of problem-matched Green's functions and therefore the efficient computation of the impedance matrix, although of primary relevance from the electromagnetic viewpoint, is only a part of the procedure necessary in order to solve the field-problem in microstrip stuctures.

II. The Electric Field Mixed Potential Integral Equation for Microstrip structures

A. MPIE Solution with the MoM and Closed Form Green's Functions

We consider N-port planar circuits, similar to that sketched in Fig. 1, with infinite transverse dimensions for both the dielectric and the ground plane; the metalization thickness is assumed negligeable.

In order to achieve improved convergence properties, we select Mixed Potential Integral Equations (MPIE) [5], [6], [7], which are solved by considering closed-form Green's functions in the spatial domain and by using the method of moments (MoM). The relative electric field integral equation is derived from the Leontovich boundary condition as:

$$\mathbf{n} \times [\mathbf{E}^e + \mathbf{E}^s] = \mathbf{Z_S}[\mathbf{n} \times \mathbf{J_S}] \qquad (1)$$

where $\mathbf{E}^e$ and $\mathbf{E}^s$ denote respectively the excitation and scattered electric field, and $\mathbf{Z}_s$ and $\mathbf{J_S}$ denote the surface impedance and electric current density respectively.

The electric field is written as a function of the vector potential $\mathbf{A}$ and the scalar potential ϕ, which satisfy the Helmholtz vector and scalar equations, respectively:

$$\mathbf{E} = -\mathrm{j}\omega\mathbf{A} - \nabla\phi \qquad (2)$$

By introducing the Green's functions $\bar{\bar{\mathbf{G}}}^{\mathbf{A}}$ and G^q for the surface electric current density $\mathbf{J_S}$ and for the surface electric charge density q_S, respectively, a Fredholm integral equation of the first kind is obtained, solvable by the MoM after suitable Green's function evaluation.

Spatial domain mixed potential Green's functions for a layered medium are expressed by Sommerfeld integrals [8] whose integrands are slowly decaying obscillating functions, hence the calculation is very time-consuming. A possible approach to circumvent this problem is the quasi-dynamic image model [9], which is not accurate enough when surface and leaky wave effects must be accounted for [10]. An ingenious device to evaluate the above mentioned Green's functions in closed form was first suggested in [1] for single-layer problems, and extended more recently to multilayer structures [2], [3]. The latter is adopted in this work.

The integral equations are solved using the Galerkin's MoM, i.e. by selecting the same functions for tests and exapnsion [11]. This way, a linear system of size N is derived from the MPIE:

$$\begin{bmatrix} Z_{xx} & Z_{xy} \\ Z_{yx} & Z_{yy} \end{bmatrix} \begin{bmatrix} I_x \\ Iy \end{bmatrix} = \begin{bmatrix} V_x \\ Vy \end{bmatrix} \qquad (3)$$

The entry Z_{ij} in the impedance matrix represents the tangential electric field generated by the j-th basis function and weighted by the i-th test one. This entry is expressed by a four-fold integral, in the spatial variables x', y' -corresponding to the source coordinates- and x, y -corresponding to the test coordinates. Part of its evaluation can be performed analytically [12] and, by paying attention to the choice of appropriate basis functions, the integrals *"can be reduced to double integrals over finite domains"* [13]. Thanks to the circular symmetry of mixed potential Green's functions, with appropriate changes into polar variables, in this work they are reduced to a simple integral in the variable r (representing the distance between source and probe):

$$\begin{aligned} Z_{xx} &= \int \left[W_{1\,x}(r)G^A_{xx}(r) - \frac{1}{\omega^2}W_{2\,x}(r)G^q(r)\right] r\,dr \\ Z_{xy} &= \int \left[-\frac{1}{\omega^2}W_{3\,x}(r)G^q(r)\right] r\,dr \\ Z_{yx} &= \int \left[-\frac{1}{\omega^2}W_{3\,y}(r)G^q(r)\right] r\,dr \\ Z_{yy} &= \int \left[W_{1\,y}(r)G^A_{yy}(r) - \frac{1}{\omega^2}W_{2\,y}(r)G^q(r)\right] r\,dr \end{aligned} \qquad (4)$$

The unknowns I_x and I_y are the (complex) amplitudes of the basis functions. The right-hand-side (rhs) vector $[V]$ depends on the excitation applied to the microstrip network.

During the simulations two sources have been used: the series voltage-source and the coaxial cable probe, which has been modelled using suited surface current distributions [14]. Using the voltage source the rhs vector is quickly filled in, as many of its terms are nulls. Both sources, voltage source and coaxial probe, give the same numerical results.

B. De-embedding Technique

The method described till now is well-suited in order to evaluate the electric current distribution on the conducting plane. Further elaborations are needed for calculating the scattering parameters of the multiport equivalent network. To this end, the latter network is analized N times, each time applying the source to a different input port and leaving open the remaining ports. Accordingly, a linear system of size N^2 is attained, exhibiting as unknowns the scattering parameters at the various ports [15]. The terminal-plane locations are chosen sufficiently far away from the discontinuities, so that only the fundamental mode propagates. In this way, we evaluate the propagation constant β_i and the complex amplitudes of the incident (a_i) and reflected (b_i) fundamental mode by considering the current samples observed at regular intervals along the center section of the line (de-embedding section), as shown in Fig. 2. The choice of an appropriate de-embedding section, and its distance from discontinuities, is performed with suitable heuristical procedures, taking into account the circuit parameters and dimensions. If the source is applied on a port, it is positioned at the end of the line, otherwise the microstrip termination is left open, and the length of this line is selected so as to allow vanishing of the evanescent modes.

The de-embedding technique is based on Prony's method [16]: the longitudinal electric current is known in $2M_s$ sample points, and is approximated with a sum of M_s complex exponentials:

$$I(x_n) \approx \sum_{m=1}^{M_s} A_m e^{\gamma_m n \Delta x} \qquad \textit{per } n{=}1,\ldots N \qquad (5)$$

The amplitudes of forward and backward waves can be evaluated from A_m terms. By applying this procedure to every line connecting the network with the remaining part of the circuit in the N possible configurations, the final linear system is built, allowing the evaluation of the network scattering parameters.

C. Code Testing

In order to demonstrate the reliability of the implemented code, we illustrate now some results which compare the theoretical analysis of some circuits. In Fig. 3 the results relative to a stub with a substrate thickness of $d = 1.27mm$ and $\varepsilon_r = 10.65$ are shown and compared with experimental results [17]. In Fig. 4 the above presented technique has been applied to characterize a microstrip double-stub discontinuity; the matching section is printed on a 10 mil substrate of relative permittivity 9.9: the magnitude and phase of the scattering parameters are compared to measurement from [18]. As illustrated, the agreement for magnitude and phase is excellent; in particular, the agreement of the phase is within 4° across the considered frequency range.

Similar results can be obtained also at a fraction of the numerical effort, by exploiting the numerical properties (i.e. the sparsity and the bandwidth reduction scheme) of the discretized matrix, as discussed in the next sections.

III. The Impedance Matrix

In the previous sections we have described the MPIE, which, discretized via MoM, is suited for the efficient simulation of arbitrarily-shaped microstrip structures. In this section, we investigate the main numerical characteristics of the approach, in order to enhance its numerical efficiency. We will point out that the use of appropriate strategies in the domain partitioning and, above all, in the solution of linear systems, when coupled with effective approaches for matrix permutation, enables considerable speed-ups without significantly affecting the accuracy of the simulation.

A. Preliminary Observations

The MPIE approach is well-suited to simulate circuits with arbitrary shapes, thanks to the partitioning of the microstrip lines into elementary rectangular cells. The simulation accuracy can also be increased by reducing the size of the elementary cells, so as to better approximate the metalization contour. A quite significant number of basis functions are needed in such a case, with a consequent increase of the computing time needed to evaluate the system matrix and to solve the MoM system (3). These two steps (system generation and solution) do not generally require the same amount of time: the linear system solution is often much more time-demanding, its complexity depending on the problem size N_s as a function N_s^α, with $2 < \alpha < 3$; on the contrary, the matrix evaluation has a linear dependence with the number of unknowns N_s.

The matrix entries' computation has been enhanced by exploiting some geometrical properties of the problem. Thanks to the radial symmetry of Green's functions, several matrix entries are identical; in addition the matrix has a particular pattern, similar to a Toeplitz one. Moreover, the electromagnetic interaction between basis and test functions generally decreases for increasing distances; their reaction term is significantly smaller than those appearing in the main diagonal. Therefore, a threshold distance value, d_c, can be found so that all the terms corresponding to the interactions for a distance greater than d_c can be omitted.

B. Properties of the System Matrix

In order to enhance the system solution time, as detailed in the following section, it has been quite useful to investigate the numerical properties of the system matrix entries in (3). Some examples are proposed (Figures 5,6), showing the matrix patterns (viz. the zero-non-zero structure) resulting for different circuit layouts. From Figures 5,6 it is also apparent where the most significant entries (with higher values) are positioned in the matrix; in these figures, the gray levels are related to the magnitude of the matrix entry: a black entry has a larger absolute value than a gray one, while values smaller than 10^{-9} have always been omitted.

Figures 5,6 clearly show that generally a few entries are significantly larger than the remaining ones, hence retaining an amount of information significant enough in order to accurately solve the problem. Therefore, the basic idea of this numerical analysis is the evaluation of the effects of a thresholding action on the matrix entries. We hence consider the following question: *for a fixed threshold v_t, so that every entry smaller than v_t is neglected, how is the numerical accuracy perturbed*? As we will see in the following, the answer to the question above will pave the way to the introduction of a bandwidth reduction scheme which substantially decreases the system solution time.

As observed from Figures 5,6, when increasing the threshold value v_t, the percentage of "zero" entries (i.e. the *sparsity* S) grows up, the non-zero positions actually depending on the circuit shape. More specifically, they are a consequence of the numbering scheme used to order the basis functions.

However, also other parameters affect the system matrix pattern: the frequency, the substrate thickness, the dielectric constant, and the elementary cell's size. We evaluate the percentage of "significant entries" in the system matrix at the highest operating frequency, which is the most critical one.

As shown in the tables in Figures 5,6, for different v_t the sparsity S of the system matrix has been computed, as well as the relative error on the system solution, and the consequent maximum error on the propagation constants and on the scattering parameters. These computations have been performed with a 7-decimal-digit notation. It can be observed that the propagation constant, being a variational quantity, is almost not affected by the thresholding action. In other words, Prony's method is more robust in the exponent evaluation, rather than in the evaluation of the complex amplitudes of the exponential functions. Finally, it can also be noticed that a direct relationship exists between v_t and the error in the system solution, whilst the link between v_t and the error affecting the scattering parameters is less straightforward. Anyway, with a threshold $v_t = 10^{-5}$, the maximum observed error on the scattering parameters is generally less than than 3%, therefore comparable with experimental errors. For such a v_t, the sparsity is generally $70\% < S < 85\%$, depending however on the microstrip circuit topology, and on the physical parameters (frequency, substrate thickness and permittivity).

IV. Bandwidth Reduced Solution of the Linear System

The analysis performed in the previous sections demonstrates that the linear system (3) is not necessarily dense: when an appropriate threshold v_t is chosen, so that the system matrix entries smaller than v_t are neglected, the matrix pattern becomes sparse. Of course, this implies some rounding errors, which, nonetheless, allow to achieve a substantial reduction of the simulation time.

A. The Impedance Matrix Sparse System Solution

A sparse system can be solved by using iterative methods (such as biconjugate-gradient algorithms), or direct sparse solvers, which are not always robust. The latter perform an LU-factorization and a backsubstitution solution; dealing with sparse matrices, this can lead to the so-called "fill-in" problem [19], i.e. the risk of obtaining dense LU factors, with a high degradation of performance. This problem is not encountered using banded direct solvers, i.e. solvers suited to systems with a banded matrix. Therefore, the effective choice for an efficient and robust solution of the system is either iterative algorithms or banded direct solvers.

The numerical complexity of both the above mentioned approaches is well-known, and is $o(NZ)$ for iterative methods (with NZ equal to the number of nonzero entries in the matrix) and $o(BN^2)$ for banded algorithms (with BN equal to the matrix bandwidth) [19]. Therefore, a straight implementation of a banded solver is generally less efficient than using an iterative solver: the system matrix, as observed in the previously shown example, generally exhibits a large bandwidth.

B. Bandwidth Reduction of the Impedance Sparse Matrix

It is worthwhile to note that some manipulations can be performed on the matrix pattern by using straightforward matrix algebra; in fact, the linear system

$$\mathbf{Ax} = \mathbf{B} \tag{6}$$

has the same numerical properties of the transformed system

$$(\mathbf{P^T A P})(\mathbf{P^T x}) = \mathbf{P^T B} \tag{7}$$

where $\mathbf{P}$ is a permutation matrix. This new system (7) has an interesting property: depending on the appropriate choice of the permutation matrix $\mathbf{P}$, it can be transformed into one with a minimum bandwidth. Once the optimum permutation matrix is found, the efficiency of the linear system solver is considerably enhanced by using a banded direct solver, and orders of magnitudes of speed-ups can be observed.

Moreover, an efficient use of direct banded solver is attractive for two more reasons: in practical applications, when we need to evaluate the dispersion curve at several frequency points, the optimum permutation as evaluated at the maximum frequency in the working range is also well-suited to every other frequency; secondly, when the same system has to be solved for different rhs, the factorization step can be performed just once, and only the backsubstitution step is repeated for every different rhs, this making direct solvers preferrable to iterative ones.

C. Methods for Bandwidth Reduction

In the past several different approaches have been presented for the evaluation of the optimum numbering of rows and columns of the $\mathbf{A}$ matrix so as to attain a minimum bandwidth [19], [20], [21], [22]. One of the most effective approaches specifically devoted to bandwidth reduction is that derived from the Cuthill-McKee (CM) method [20], [21].

The main idea of this class of algorithms is related to the graph representation of the matrix. Let us consider the matrix$\mathbf{A}$ with a symmetric zero-non-zero pattern . A so-called "incidence" graph can be easily associated to the latter matrix, as in Fig. 7, where each row/column is represented by a node, and nodes i and j are connected each other if and only if the entry a_{ij} is not a null. The incidence graph can be partitioned into "levels", as shown in the same picture. All the nodes with the same "distance" from the graph's root are included in the same level. It can be intuitively understood (and this is demonstrated in [23]) that the larger the number of levels in an incidence graph, the smaller the bandwidth of the matrix. Therefore, the goal is to find out a node numbering scheme that maximizes the number of levels in the incidence graph.

The basic items of bandwidth reduction algorithms, while being quite well-known to those acquainted with bandwidth reduction and graph partitioning, are not trivial, and can be summarized as:

- partitioning phase: select a new root node for the incidence graph, and cut some edges, such that there are edges only between nodes belonging to the same level, or to two adjacent levels
- numbering phase: number the nodes by increasing level, and inside each level number them according to a particular criterion.

Starting from these ideas, a new approach (called WBRA) has been proposed by the authors in [24], [25], implementing some improved features and enhancing the performance of bandwidth reduction and computing times quite substantially. The WBRA's main new features can be summarized in the following way:

- partitioning phase: the graph is divided into levels so that the minimum number of nodes belong to the same node (in fact, the bandwidth is directly affected by the size of the largest subset). Differently from the CM approach, the WBRA approach introduces some heuristic criteria specifically devoted to this goal
- numbering phase: the WBRA algorithm applies the numbering to a set of "promising" level structures determined during the partitioning phase. Some advanced (and rather complex) heuristic criteria are then used to select the best level partitioning and numbering in an efficient way. Differently from CM approach, WBRA explores different level structures, this allowing better performance.

A simple example of the final matrix, level structure and incidence graph for a sparse matrix is shown in Fig.

8. It can be easily observed that, starting from the incidence graph in Fig. 7, if node 3 is transformed into node 1, node 1 into 2, 4 into 3, and 2 into 4, the graph of Fig. 8 and its companion matrix are obtained. The corresponding P is

$$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

V. Results

We present numerical results demonstrating the high-performance of the approach for two cases: a 2-port circuit (a double stub), and a 4-port one (a branch coupler). All the reported simulations have been run on an entry level workstation, with a 32MB RAM. The use of a strategy for reducing the numerical complexity of the problem (the thresholding), coupled with a state-of-the-art method for bandwidth reduction of the system matrix in (3), highly enhance the performance of the code. The results compare the performance of the approach in three different possible scenaries:

- No thresholding is performed, a dense solver (Gauss-Jordan, GJ) is used in the (3)
- Thresholding is performed with a fixed v_t, and an iterative sparse solver (biconjugate-gradient, BCG) is used
- Thresholding is performed with a fixed v_t, and the bandwidth of the system matrix minimized using WBRA. A banded solver (BN) is then used in the (3)

All the three linear system solvers (GJ, BCG, and BN) are selected from SLATEC public-domain collection. The iterative sparse solver is generally used with 100-180 iterations, so that an error of 10^{-9} is achieved (this value is enough to guarantee an appropriate convergence also when a thresholding is performed). The threshold value, as discussed in section 6.2, is 10^{-5}.

The computation is divided into two main tasks: the time necessary to evaluate the matrix and the rhs in the (3) (we call this step "system generation"), and the time to solve the same system. When the GJ approach is used, every frequency point is simulated with the same computing time. In the BCG and BN case, this is not true. The first frequency point (the upper limit in the frequency range) is used to determine the matrix entries which are smaller than v_t, and need not to be computed at next frequency steps. Moreover, in the BN case, also the permutation matrix **P** so that the bandwidth is reduced, is computed, and the system rearrangement performed as suggested in (7). Therefore, in the BCG and BN case, the first frequency point analysis is slower than the following ones.

A. 2-Port Network

The two-port network represented in Fig. 5 has been simulated in the range 2.5-3.5 GHz. The dielectric constant is 2.6, and the cell dimension is 3mm (uniform meshing with square cells). The system size in (3) is 220. In Table II we show the performance results for the main tasks. Times to perform the tasks (system generation, solution, and bandwidth reduction) are given in seconds. An IBM RS6000 250 T, an entry level workstation, has been used. Times in Table I refer to a single frequency point. Times for the system generation for the BCG and BN case suppose that the thresholding on the matrix entries has already been performed. As previously discussed, this can not be assumed for the first frequency point. In that case, the system generation is performed in 53 s for the GJ case, and in 44 s in the BCG and BN cases.

Table I: Times for the different tasks (in seconds) for one frequency point.

Method	*System Gen*	*System Sol.*	*Bandwidth .Red.*
GJ	53	4.5	-
BCG	11.9	3	-
BN	11.9	0.2	0.5

The sparse matrix solved with BCG is obtained with a 10^{-5} threshold, and its bandwidth is 208. Using WBRA it is reduced to 72. It must also be observed that the system (3) solution is repeated for each frequency point as many times as the number of ports, in order to evaluate the scattering parameters.

In Table II the timings for a complete frequency analysis (100 frequency points) are presented for the three methods. Remember that if the first frequency point is the upper-frequency one, the bandwidth minimization can be performed only once (as previously discussed in section 6.2).

Table II: Times (in seconds) for 100 frequency point.

Method	*Upper Freq. Point*	*Remaining 99 Freq. Points*	*Total Time*
GJ	62	6138	6200
BCG	50	1772	1832
BN	45	1198	1243

It can be seen that a global speed-up of more than 3 times is attained with the BCG strategy, and of around 5 with the BN+WBRA strategy. This is due to the huge performance improvement obtained in the linear system solution. Using the thresholding, the BCG solves the sparse system in around 3s instead of 4.5s (dense GJ). The BN solver, after the bandwidth minimization (which takes 0.5 s) performs the solution in 0.2s. Moreover, the banded solver is a direct LU algorithm. As stressed before, for each frequency point the system is solved as many times as the ports are (this case 2 times) with different rhs. This means that using the banded solver the only backsubstitution step is performed twice, and it is quite well known that this step is very fast, compared with the LU factorization step.

The improvement in the efficiency is obtained without degradating too much the accuracy of the simulation: both the BCG and the BN strategy affect the result with an error of around 3%, as already shown in the table in Fig. 5. This is quite satisfactory, and still comparable with experimental errors.

B. 4-Port Network

The results for the 4-port coupler of Fig. 6 are presented following the same logical path of the previous section: the GJ, BCG and BN strategy are compared, demonstrating the superior performance of the BN+WBRA strategy. As the number of ports is increased in this example, very substancial speed-ups are achieved.

The circuit is studied in the range 2.5-3.5 GHz, with a substrate with $\varepsilon = 2.6$, and square cell size of edge 3mm. In Table III results are shown, with times in seconds referred to the above mentioned IBM 250 T. Data do not refer to the first frequency point (upper limit of the range). In that case, the system is generated in 111.4 s with the GJ approach, and in 81 s with the BCG and BN one. The system size is 401.

Table III: Times for the different tasks (in seconds) for one frequency point.

Method	*System Gen.*	*System Sol.*	*Bandwidth Red.*
GJ	111.4	57	-
BCG	15.4	7.4	-
BN	15.4	0.6	2

In this case, the bandwidth of the sparse matrix obtained performing a thresholding with $v_t = 10^{-5}$ is 310, and by applying the WBRA it is reduced to 82, with a huge enhancement in the solution time when the BN solver is used. For the coupler, the speed-ups for a 100-point frequency curve are very interesting, as the system solution is repeated 4 times for each frequency point, and a direct LU banded solver is extremely suited and effective. Table IV demonstrates this, showing the total times for the whole frequency analysis:

Table IV: Times (in seconds) for 100 frequency point.

Method	*Upper Freq. Point*	*Remaining 99 Freq. Points*	*Total Time*
GJ	338.6	33521	33860
BCG	141	4356	4497
BN	112.2	1723	1835

It can be seen that the BCG strategy is more than 7 times faster than the standard implementation of the MPIE approach solved with the MoM. A strategy based on WBRA for bandwidth reduction and a BN solver is more than 18 times faster. Also in this case, the error in the solution is around 3% (see the table in Fig. 6) on the computed scattering parameters.

This last example demonstrates that on increasing the number of ports, and the complexity of the circuit layout, the proposed strategy becomes more and more high-performing.

VI. Conclusions

An efficient electromagnetic and numerical approach has been proposed for the analysis of arbitrarily-shaped microstrip circuits, based on the solution of the Mixed-Potential Integral Equations via Method of Moments. The use of analytically evaluated closed-form Green's functions and general de-embedding techniques makes the code efficient from the electromagnetic view-point.

A detailed analysis of the resulting linear system, which solution represents one of the key-issues in the code performance, has demonstrated that, even for standard rooftop basis functions, at the cost of small degradation of the solution accuracy, the generally dense linear system can be reduced to a sparse one, with a sparsity generally in the range 70-85%.

The use of a method for the bandwidth reduction of sparse matrices developed by the authors, while outperforming the previously proposed approaches for bandwidth reduction, when coupled with a banded direct solver, is shown to produce a substantial speed-up in the numerical simulation of microstrip circuits. A speed-up of 5 times has been achieved on a 2-port circuit, and of 18 times on a 4-port circuit.

References

[1] Y. L. Chow, J. J. Yang, D. G. Fang and G. E. Howard, "A closed-form spatial Green's function for thick microstrip substrate", *IEEE Trans. Microwave Theory Tech.*, vol. 39: pp. 588–592, Mar. 1991.

[2] M. I. Aksun and R. Mittra, "Derivation of closed-form Green's functions for a general microstrip geometry", *IEEE Trans. Microwave Theory Tech.*, vol. 40: pp. 2055–2062, Nov. 1992.

[3] G. Dural and M. I. Aksun, "Closed-form Green's functions for general sources and stratified media", *IEEE Trans. Microwave Theory Tech.*, vol. 43: pp. 1545–1552, Jul. 1995.

[4] T. Rozzi and M. Mongiardo, *"Open electromagnetic waveguides"* , IEE, London, 1997.

[5] J. R. Mosig and F. E. Gardiol, "Analytical and numerical techniques in the Green's function treatment of microstrip antennas and scatterers", *Proc. Inst. Elec. Eng., pt. H: Microwave Optics Antennas*, vol. 130: pp. 175–182, 1983.

[6] J. R. Mosig and F. E. Gardiol, "General integral equation formulation for microstrip antennas and scatterers", *Proc. Inst. Elec. Eng., pt. H: Microwave Optics Antennas*, vol. 132: pp. 424–432, Dec. 1985.

[7] J. R. Mosig, "Arbitrarly shaped microstrip structures and their analysis with a mixed potential integral equation", *IEEE Trans. Microwave Theory Tech.*, vol. 36: pp. 314–323, Feb. 1988.

[8] A. Sommerfeld, *"Partial Differential Equations in Physics"*, Academic Press, New York, 1949.

[9] Y. L. Chow, "An approximate dynamic spatial Green's function in three dimensions for finite length microstrip lines",*IEEE Trans. Microwave Theory Tech.*, vol. 28: pp. 393–397, 1980.

[10] J. R. Mosig and F. E. Gardiol, "A dynamical radiation model for microstrip structures". In *Advanced in Electronics and Electron Physics*, vol. 59: pp. 175–182. London Academic Press, 1982.

[11] R. F. Harrington, *"Field Computation by Moment Method"*, Macmillan, New York, 1968.

[12] L. Barlatey, J. R. Mosig and T. Sphicopoulos, "Analysis of stacked microstrip patches with a mixed potential integral equation", *IEEE Trans. Microwave Theory Tech.*, vol. 38: pp. 608–615, May 1990.

[13] M. I. Aksun and R. Mittra, "Estimation of spurious radiation from microstrip etches using closed-form Green's functions", *IEEE Trans. Microwave Theory Tech.*, vol. 40: pp. 2063–2070, Nov. 1992.

[14] J. R. Mosig, "Integral Equation Technique". In *Numerical Techniques for Microwave and Millimeter Wave Passive Structures*, chapter 3. T. Itoh, New York, 1986.

[15] I. Park, R. Mittra and M. I. Aksun, "Numerically efficient analysis of planar microstrip configurations using closed-form Green's functions", *IEEE Trans. Microwave Theory Tech.*, vol. 43: pp. 394–400, Feb. 1995.

[16] R. C. Hall and J. R. Mosig, "The analysis of arbitrarly shaped aperture-coupled patch antennas via a mixed-potential integral equation", *IEEE Trans. Antennas Propagat.*, vol. 44: pp. 608–614, May 1996.

[17] R. Jackson, "Full-wave finite element analysis of irregular microstrip discontinuities", *IEEE Trans. Microwave Theory Tech.*, vol. 37: pp. 81–89, Jan. 1989.

[18] W. P. Harokopus and P. B. Katehi, "Characterization of microstrip discontinuities on multilayer dielectric substrates including radiation losses ", *IEEE Trans. Microwave Theory Tech.*, vol. 37: pp. 2058–2066, Dec. 1989.

[19] I. S. Duff, A. M. Erisman and J. K. Reid, *Direct methods for sparse matrices*. Oxford University Press, 1986.

[20] E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices", in *ACM National Conference*, New York 1969.

[21] N. E. Gibbs, W. G. Poole and P. K. Stockmeier, "An algorithm for reducing the bandwidth and profile of sparse matrix", *SIAM Journal of Numerical Analysis* vol. 13(2): pp. 236–250, 1996.

[22] G. Dueck and J. Jeffs, "A heuristic bandwidth reduction algorithm". *Journal of Combinatorial Mathematics and Combinatorial Computing*, vol 18: pp. 97–108, 1995.

[23] D. Kuo and G. J. Chang, "The profile minimization problem in trees", *SIAM Journal on Computing* vol. 23(1): pp. 71–81, 1994.

[24] A. Esposito, S. Fiorenzo Catalano, F. Malucelli and L. Tarricone, "Sparse matrix bandwidth reduction: algorithms, applications and real industrial sases in electromagnetics", in *High performance Algorithms for Structured Matrix Problems*, ABLEX , Norwood, NJ 1997 (forthcoming).

[25] A. Esposito, S. Fiorenzo Catalano, F. Malucelli and L. Tarricone, "A wonderful bandwidth matrix reduction algorithm", to appear in *Operations Research Letters*, 1997.

[26] M. I. Aksun and R. Mittra, "Choices of expansion and testing functions for the method of moments applied to a class of electromagnetic problems", *IEEE Trans. Microwave Theory Tech.*, vol. 41: pp. 503–508, Mar. 1993.

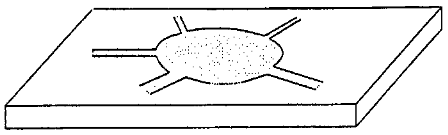


Fig. 1. A general microstrip structure.

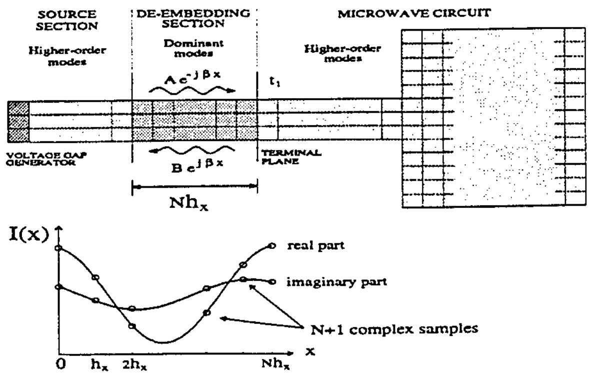


Fig. 2. De-embedding section.

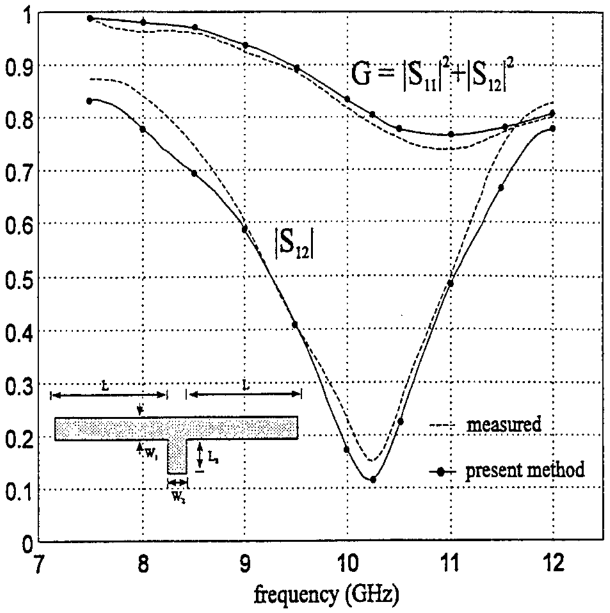
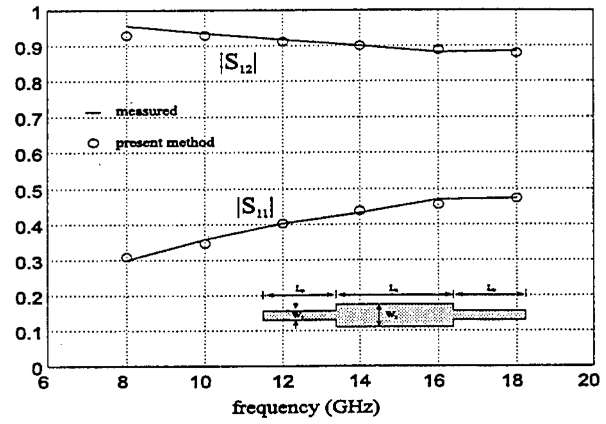


Fig. 3. Scattering parameters for microstrip stub ($W_1 = W_2 = 1.44mm$, $L = 2.16mm$, $\varepsilon_r = 10.65$, $h = 1.27mm$)



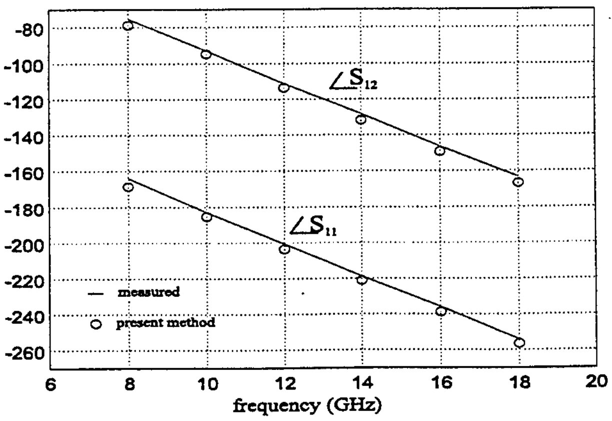
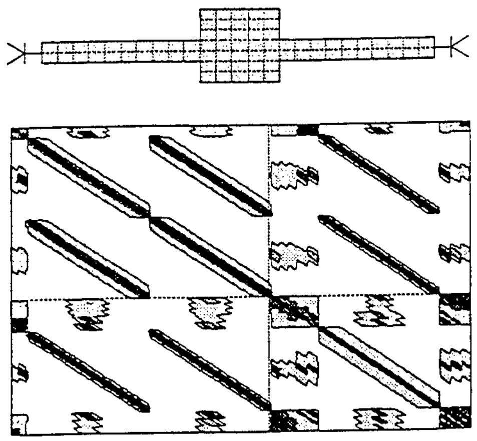


Fig. 4. Magnitude and phase of the scattering parameters for microstrip double-stub ($L_P = 30mil$, $L_S = 50.6mil$, $W_1 = 9.2mil$, $W_2 = 23mil$, $\varepsilon_r = 9.9$, $h = 10mil$)



v_t	S	Solution Error	β Error	Scatt. Par. Error
10^{-7}	14.64 %	0.00 %	0.02 %	0.55 %
10^{-6}	35.97 %	0.02 %	0.05 %	0.96 %
10^{-5}	72.94 %	5.04 %	0.04 %	0.98 %
10^{-4}	85.71 %	5.61 %	0.07 %	1.95 %
10^{-3}	91.09 %	62.23 %	0.27 %	7.59 %
10^{-2}	92.7 %	212.64 %	3.17 %	22.58 %

Fig. 5. The layout of a double-stub, and the corresponding matrix pattern for different v_t values. In the table, the sparsity, solution error, and error in β and scattering parameters are shown for different v_t values.

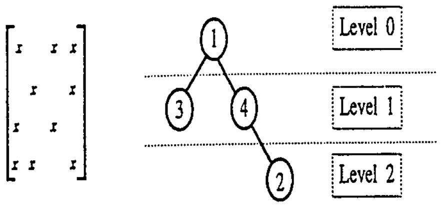
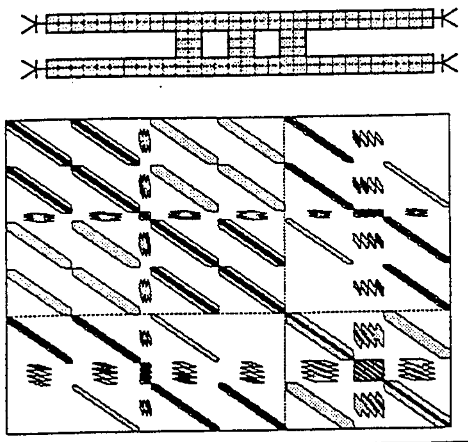


Fig. 7. Incidence graph and level partitioning for a given matrix.



v_t	S	Solution Error	β Error	Scatt. Par. Error
10^{-7}	10.84 %	0.00 %	0.01 %	0.10 %
10^{-6}	35.41 %	0.07 %	0.04 %	0.24 %
10^{-5}	80.74 %	4.92 %	0.12 %	3.02 %
10^{-4}	93.61 %	8.14 %	0.50 %	3.83 %
10^{-3}	95.73 %	16.59 %	0.83 %	10.09 %
10^{-2}	96.55 %	143.03 %	3.71 %	34.31 %

Fig. 6. The layout of a branch-line coupler, and the corresponding matrix pattern for differentv_t values. In the table, the sparsity, solution error, and error in β and scattering parameters are shown for different v_t values.

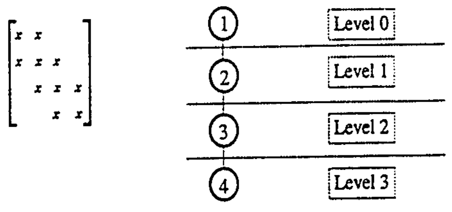


Fig. 8. Reordered matrix, corresponding incidence graph and level partitioning.

Speedup Using a Modal Frequency Method for Finite Element Analysis of a Dual–Mode Microwave Filter

John R. Brauer
Ansoft Corporation
929 N. Astor Street, Suite 506, Milwaukee, WI 53202 USA
emails: jbrauer@execpc.com, brauer@ansoft.com, brauer@msoe.edu

Abstract—**Computer time required for finite element analysis of microwave filters is reduced by more than an order of magnitude by using modal frequency rather than direct frequency methods. In the conventional direct frequency method, the number of unknowns is equal to the number of edge degrees of freedom. Instead, the new modal frequency method first computes the 3D modes and then uses them as basis functions, thereby greatly reducing the number of degrees of freedom. The two methods are applied to the European benchmark problem of a dual–mode microwave filter. The modal frequency method obtains essentially the same results as the direct frequency method, but when analyzing 201 frequencies it yields a speedup factor of 15.**

INTRODUCTION

Microwave resonators are often used to make filters. If the coupling coefficients are small and the quality factor Q is high, then narrow passbands or stopbands may require analysis at a hundreds of frequencies. In conventional direct frequency finite element analysis (FEA), total solution time is directly proportional to the number of frequencies analyzed, and may therefore be hundreds of times longer than the solution at one frequency.

To reduce computer time, several methods of reduced order modeling have been used. Asymptotic waveform analysis (AWE) has been used for several years [1], and it has been successfully combined with finite element analysis [2]. The most recent method of AWE is called PVL for Padé via Lanczos process [3]–[6]. While PVL can extend the frequency range compared to standard AWE methods, as of now no AWE method is guaranteed to find all resonances over a specified frequency range.

In a new technique called modal frequency FEA [7], a 3D real Lanczos eigenvalue analysis with Sturm sequencing is first performed to reliably find *all* resonances of low loss devices. The resulting eigenvectors are used as basis functions for solutions over a range of frequencies, thereby possibly saving computer time if S–parameters are needed for a large number of frequencies.

This paper begins with a review of the modal frequency method of FEA and some of its recent applications. Then the new modal frequency method is applied to a benchmark problem from a European magazine. The problem is a two–port dual–mode cylindrical six–cavity filter with iris coupling to rectangular waveguides. The modal frequency results will be compared with those obtained by the conventional direct frequency method and by measurements.

MODAL FREQUENCY FINITE ELEMENT ANALYSIS

Conventional direct frequency FEA consists of solving the complex matrix equation [7], [8] with angular frequency ω:

$$[\,-\omega^2[M] + j\omega[C] + [K]\,]\{u\} = \{P\} \qquad (1)$$

where [M] is the permittance matrix (proportional to permittivity), [C] is the conductance matrix (proportional to conductivity), and [K] is the reluctance matrix (inversely proportional to permeability). For the 3D edge finite elements used here, the unknown vector {u} consists of edge magnetic vector potentials $\overline{A}$. The electric field is then $-j\omega\overline{A}$. {P} is the excitation vector, which for S–parameter computations is located at the ports. The {u} vector has as many degrees of freedom as there are finite element edge unknowns, which usually number in the tens of thousands. Note that the left hand matrix changes with frequency and thus solution time is proportional to the number of frequencies analyzed.

Instead of solving (1) directly, we can first compute the real eigenvalues and eigenvectors, denoted by $\{\phi_i\}$, of the 3D finite element model. Then a *modal frequency* solution is assumed to be a linear combination of the eigenvectors, expressed as [7]:

$$\{u\} = [\phi]\{q\} \qquad (2)$$

where the matrix $[\phi]$ is made up of m columns of individual orthogonal eigenvectors $\{\phi_i\}$, and the vector {q} contains all of the coefficients. If there are n direct degrees of freedom in a problem (the length of the column vector {u}), then $[\phi]$ is an (n x m) matrix. This transformation can be highly accurate when all n eigenvectors of the system are used. In many cases only a small approximation is introduced if a limited number of eigenvectors in a specified frequency range is used.

The frequency range of the eigensolution should include all modes that are expected to be excited. All of the real modes over any finite frequency range are rigorously computed in our software [7] using a Sturm sequenced Lanczos algorithm. Then the final solution is obtained by substituting (2) into (1):

$$-\omega^2[M][\phi]\{q\} + j\omega[C][\phi]\{q\} + [K][\phi]\{q\} = \{P\} \quad (3)$$

Premultiplying both sides by $[\phi]^T$ results in:

$$-\omega^2[\phi]^T[M][\phi]\{q\} + j\omega[\phi]^T[C][\phi]\{q\} + [\phi]^T[K][\phi]\{q\} = [\phi]^T\{P\} \quad (4)$$

where the three new *modal* matrices are:

$$[m] = [\phi]^T[M][\phi] \quad (5)$$
$$[c] = [\phi]^T[C][\phi] \quad (6)$$
$$[k] = [\phi]^T[K][\phi] \quad (7)$$

Thus (4) can be rewritten as the *modal frequency equation*:

$$(-\omega^2[m] + j\omega[c] + [k])\{q\} = \{p\} \quad (8)$$

Recall that the unknown $\{q\}$ vector is of length equal to the selected number of real 3D modes. This reduction in the number of unknowns from those of $\{u\}$ of (1) can often lead to a substantial computational speedup compared with direct frequency FEA.

Because real modes are assumed, the modal frequency method is applicable only to low loss problems. While dielectric and ferrite material losses can be analyzed directly, wall losses can only be analyzed indirectly using equivalent lossy air or other filler material. Also, real modes cannot represent radiation boundaries, and thus antennas cannot be analyzed by modal frequency FEA.

A recent paper [7] describes the theory of modal frequency FEA in detail. It includes a discussion of S–parameter computations using approximate port boundary conditions.

The same paper also applies modal frequency FEA to computing the S–parameters of two filters. One filter is a cutoff–coupled rectangular dielectric resonator filter, and the other is a coax–fed rectangular box containing cylindrical dielectric resonators. Modal frequency FEA results obtained are similar to those obtained by direct frequency and by measurements. Compared with direct frequency FEA, modal frequency FEA obtained speedup factors ranging from 1.4 to 4 for analyses at approximately 100 frequencies.

Another recent paper [9] applies modal frequency FEA to two other filters. One is the ACES/TEAM 19 single cavity filter, for which accurate S–parameters were obtained with a speedup of 10 over direct frequency FEA. The other example analyzed was an improper variant of a dual–mode filter; here the proper geometry is used in the analysis via both direct frequency and modal frequency FEA.

DESCRIPTION OF THE DUAL–MODE MICROWAVE FILTER

Dual–mode microwave filters use two similar propagating waveguide modes to produce a bandpass filter. The transmission coefficient often shows excellent rejection outside the desired passband [10].

The earliest dual–mode filters produced their dual modes by means of adjustable screws inserted in waveguide walls [11]. Recently, an improved design uses rotated apertures to produce dual couter–rotating modes in a circular waveguide [12]. Usually multiple apertures are placed along the length of the filter, where each aperture is rotated sequentially. The rotations are either all in the clockwise direction or all counterclockwise.

The dual–mode filter analyzed in this paper has elliptical apertures that are rotated sequentially in the counterclockwise direction [13]. Figs. 1 and 2 show the outside of the filter, which is made up of six cylindrical cavities fed by rectangular waveguide at both ends. The rectangular waveguide cross section is 1.905 by 0.9525 cm. The cylindrical cavities are all of diameter 2.40 cm. The geometry was entered via solid modeling commands in the preprocessing software. The filter is entirely filled with air and all of its walls are made of aluminum. Thus the filter is here assumed to be lossless.

Fig. 3 is an isometric view at the same angles of Fig.1, but showing the inside of the filter. There are seven elliptical apertures at positions and major axis angles [13] listed in Table 1. Fig. 4 is a front view of the inside of the filter, showing the ellipses. The two large tilted ellipses of Table 1 have a major axis of 2.4 cm and a minor axis of 2.1 cm. The two large verical ellipses are 2.4 by 2.0414 cm. The two end ellipses are 1.278 by 0.4 cm, and the single center ellipse is 0.87 by 0.4 cm.

TABLE 1. Elliptical apertures in filter

z min (cm)	z max (cm)	major axis angle (deg.)
0.750	0.923	180 (end ellipse)
1.4729	1.523	135 (large tilted ellipse)
2.0728	2.132	90 (large vertical ellipse)
2.6827	2.8342	90 (center ellipse)
3.3841	3.4441	90 (large vertical ellipse)
3.994	4.044	45 (large tilted ellipse)
4.5939	4.7669	0 (end ellipse)

Fig. 5 shows the measured S–parameters. They were obtained using an experimental filter that is specified to

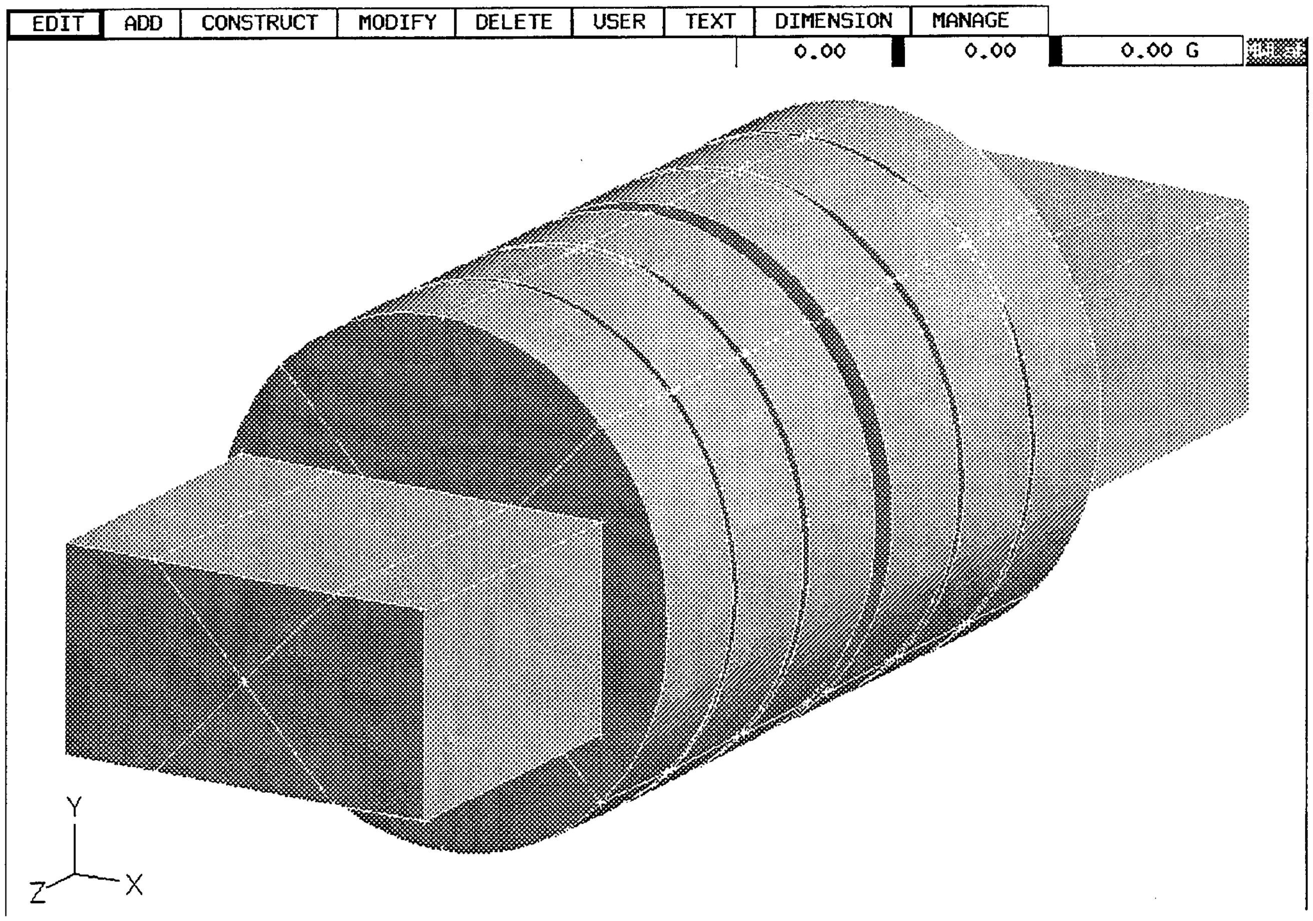


Fig. 1. Isometric view of outside of solid geometry model of dual–mode filter.

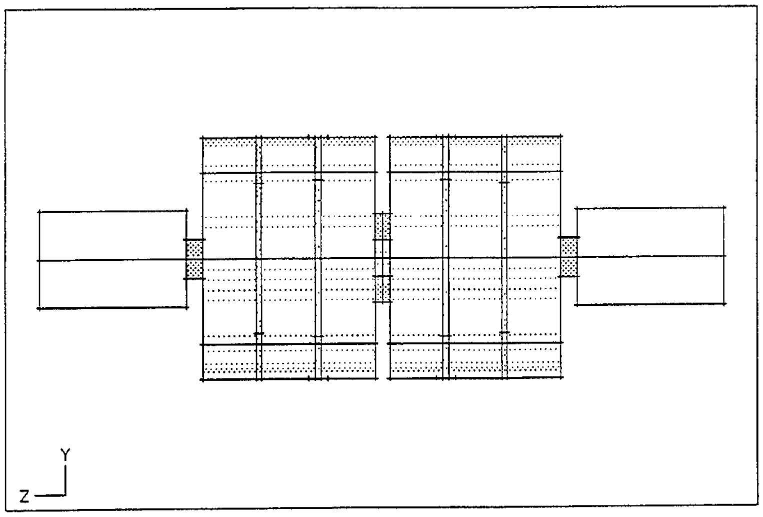


Fig. 2. Side view of dual–mode filter of Fig. 1.

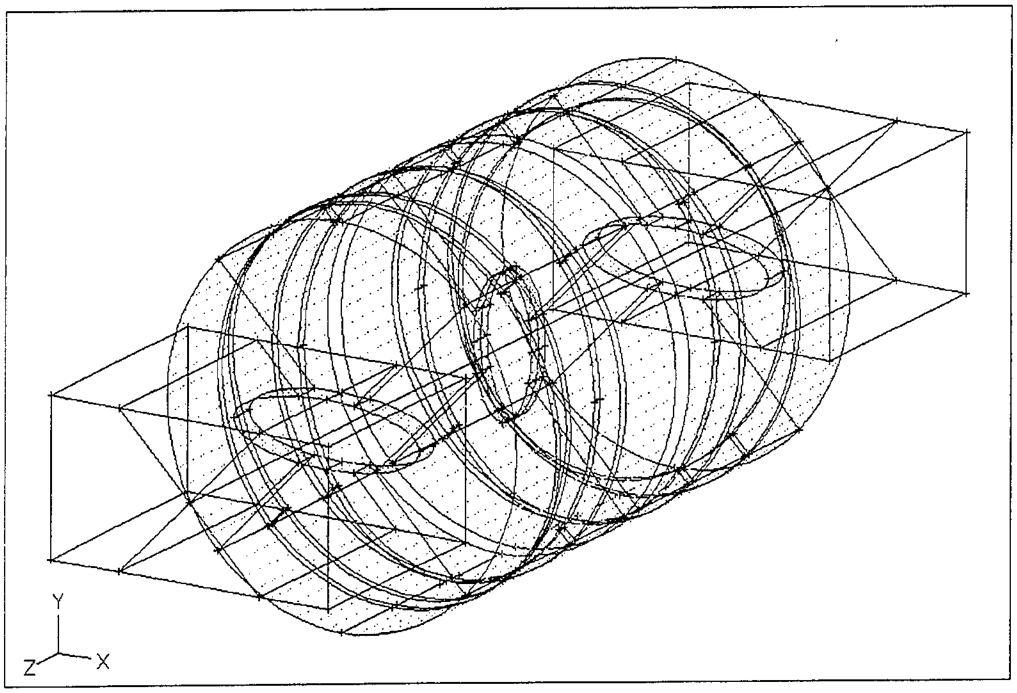


Fig. 3. Isometric view of inside of solid geometry model of dual–mode filter.

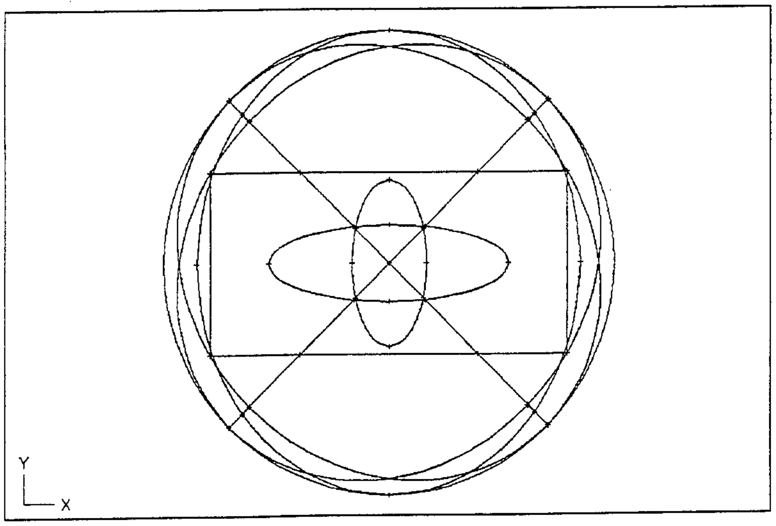


Fig. 4. Front view of inside of solid geometry model of dual–mode filter.

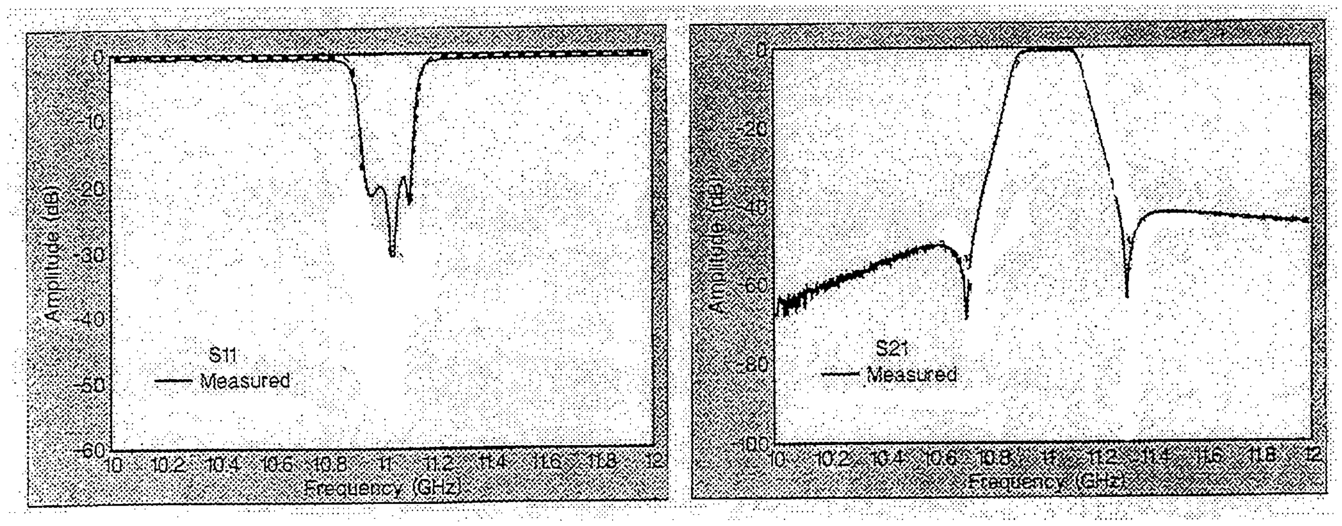


Fig. 5. Measured S–parameters of dual–mode filter [13].

obey the above dimensions within a tolerance better than plus or minus ten microns.

3D FINITE ELEMENT MODEL AND COMPARATIVE RESULTS

Even though the six cylindrical cavities of Figs. 1 through 4 are all symmetric about several mirror planes, the rotation of the elliptical aperture axes means that the filter possesses no plane of symmetry. Thus the entire filter must be modeled, here using finite elements.

Fig. 6 shows the finite element model, which consists of 10,169 second order edge (H1–curl) tetrahedrons. They have a total of 64,450 edge degrees of freedom.

The model of Fig. 6 was submitted to our software, which computed the fields and S–parameters using equations given previously [7]. The S–parameters were computed using the software's direct frequency capability of (1) and its modal frequency capability of (8) at 201 frequencies from 10 to 12 GHz, spaced by 10 MHz.

The computed S–parameters are shown in Fig. 7 for direct frequency and in Fig. 8 for modal frequency FEA.

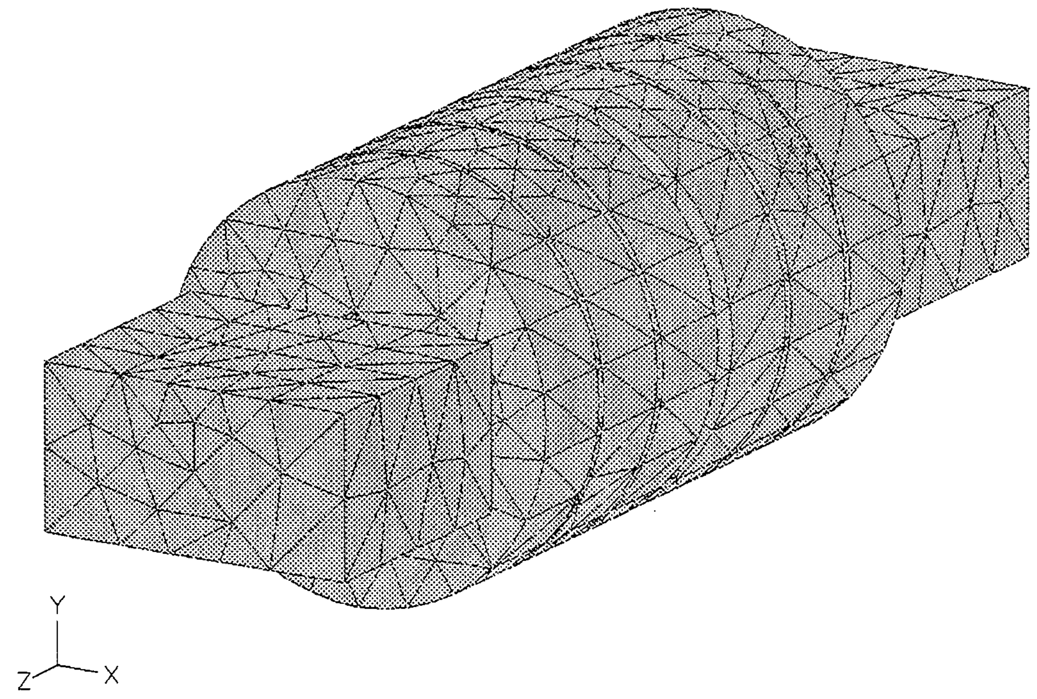


Fig. 6. Finite element model of filter, made up of 10,169 H1–curl tetrahedrons.

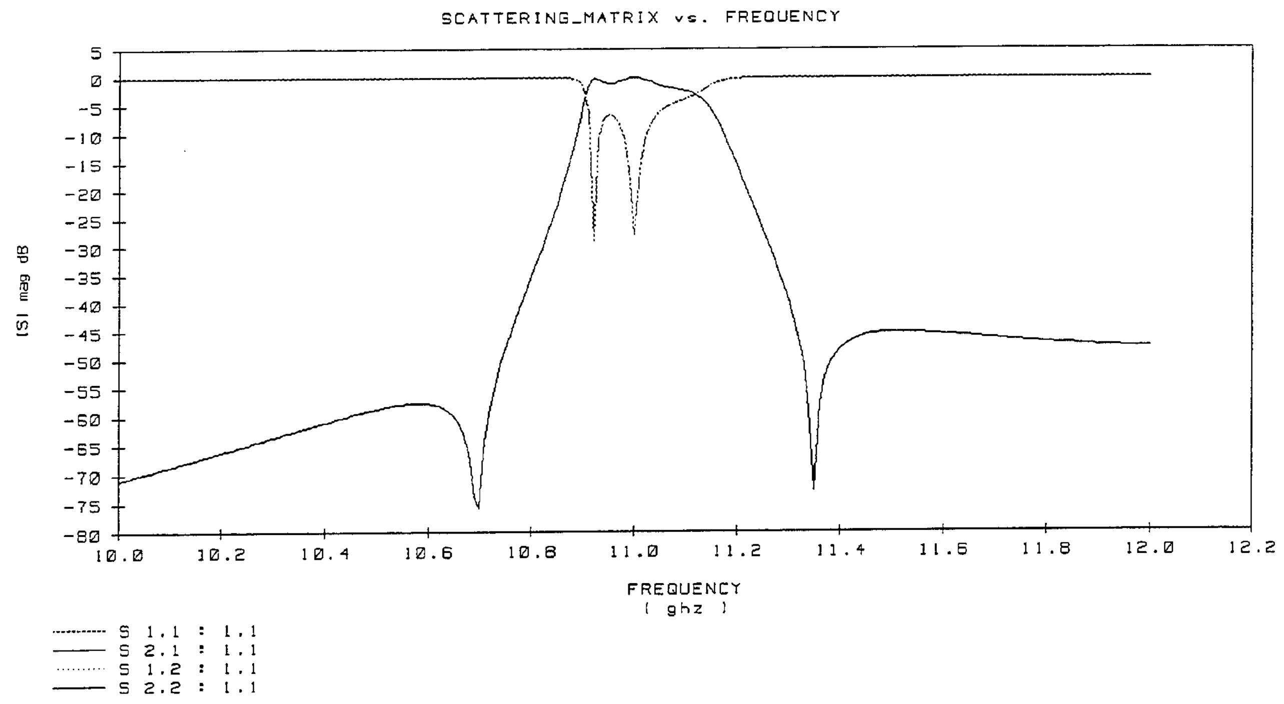


Fig. 7. S–parameters computed using direct frequency method at 201 frequencies.

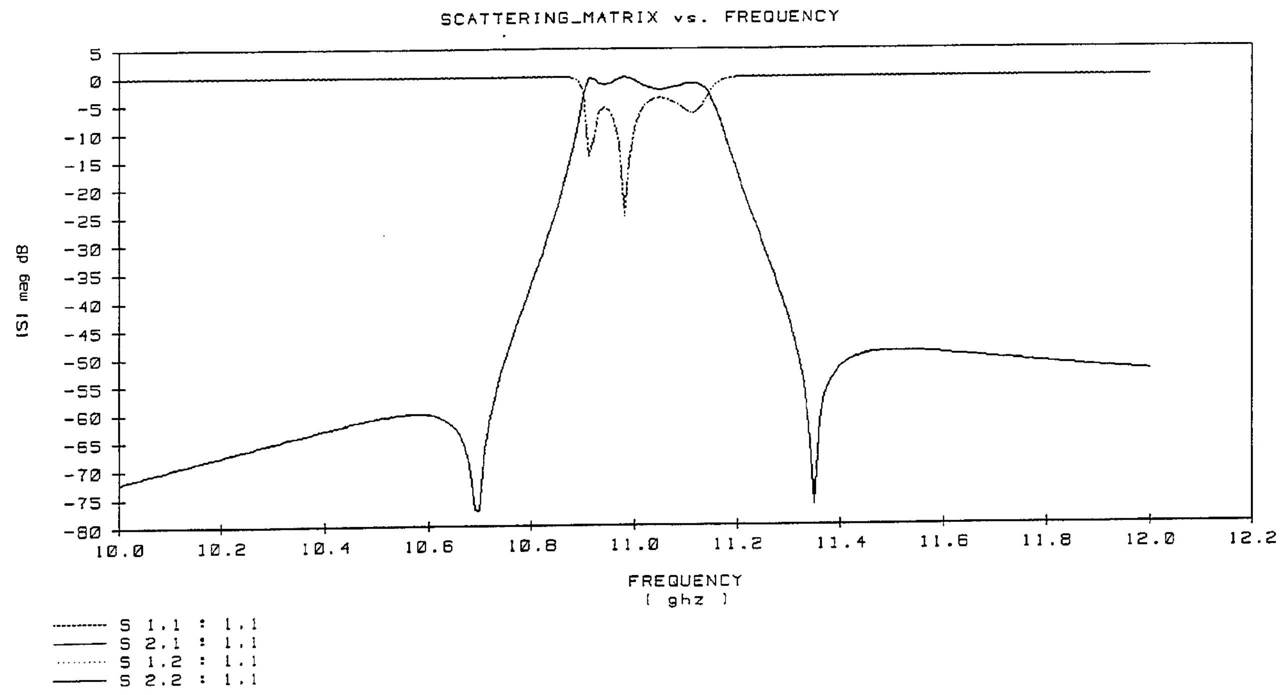


Fig. 8. S–parameters computed using modal frequency method at 201 frequencies.

Comparing Figs. 5, 7, and 8 shows that the direct and modal results are very similar, both for S11 and S21. The measured S21 of Fig. 5 appears similar to the computed S21 of Figs. 7 and 8. The measured S11 of Fig. 5 differs significantly from the computations of Figs. 7 and 8 in the passband region.

There are several possible explanations for the difference between the measured and computed S11. One is that the measurements may have been made on a filter with a greater diameter, evidently denoted as x.xxx in [13], than the 2.40 cm assumed in Fig. 1. Another possibility is that more finite elements may be needed, whether direct frequency FEA or modal frequency FEA is used. The expected accuracy of the computed and measured S parameters is on the order of plus or minus 5 db.

COMPUTER PERFORMANCE

Both the direct frequency and modal frequency computations of Figs. 7 and 8 were made on a Hewlett–Packard model 735/125 workstation. The CPU time was 40,718 seconds for the direct frequency analysis and 2,767 seconds for the modal frequency analysis. The modal frequency analysis first searched for all 3D modes from 5 to 12.5 GHz, and found ten modes at frequencies ranging from 8.27 to 11.36 GHz.

The software developed here is also available on other high–performance computers including several types of Cray parallel processors. On the Cray, a vectorized and parallelized sparse solver has been developed [14]. The software includes not only the solid geometry preprocessing of Figs. 1 through 4, but also the automatic mesh generation of Fig. 6, as well as postprocessing capabilities such as the graphs of Figs. 7 and 8.

Eigenvalue extraction and other computationally intensive tasks in the software are carried out using special vector kernels that have been optimized for various high–performance computer platforms. The most important vector kernels are usually BLAS (basic linear algebraic subroutines). Level I–type kernels include SAXPY and DOT [14], [15]. Block kernels are also specific to the particular hardware and include double kernels such as multiplications of scalars and vector blocks. BLAS level II and especially level III–type kernels are not favored in commercial finite element systems, because they are usually incompatible with simple fortran matrix storage schemes.

Recently, sparse matrix methods have been implemented that greatly enhance performance for very large finite element matrices. Sparse BLAS kernels such as SAXPI have become heavily used in sparse matrix routines. Also enhancing performance of sparse matrix processing are new resequencing methods based on minimum degree ordering [14], [16].

CONCLUSION

A dual–mode microwave filter has been analyzed by both conventional direct frequency FEA and by a new modal frequency FEA technique. Both methods have obtained S21 at 201 frequencies that agrees well with measurements, but their S11 disagrees somewhat with measurements. While the modal frequency method obtained essentially the same S–parameters as the direct frequency method, it has achieved a speedup factor of 15. Thus modal frequency FEA is attractive for analysis of filters and other low–loss resonant microwave devices.

ACKNOWLEDGMENTS

The software developed here is available commercially as MicroWaveLab^{TM} from Ansoft Corporation, Four Station Square, Suite 660, Pittsburgh, PA 15219, USA.

The author thanks Dr. Zoltan Cendes and Dr. Nancy Lambert of Ansoft Corporation, and Professor Ray Palmer of the Milwaukee School of Engineering, for their kind support.

REFERENCES

[1] E. Chiprout and M. S. Nakhla, *Asymptotic Waveform Evaluation*, Norwell, MA: Kluwer, 1994.

[2] Din Sun, John Manges, Xingchao Yuan, and Zoltan Cendes, “Spurious modes in finite element methods,” *IEEE Antennas and Propagation Magazine*, v. 37, Oct. 1995, pp. 12–24.

[3] William T. Smith, Rodney D. Slone, and Sudip K. Das, “Recent progress in reduced–order modeling of electrical interconnects using asymptotic waveform evaluation and Padé via Lanczos process,” *Applied Computational Electromagnetics Society Newsletter*, v. 12, July 1997, pp. 46–71.

[4] Din–Kow Sun, “ALPS– an adaptive Lanczos–Padé approximation for the spectral solution of mixed–potential integral equations,” *URSI Radio Science Meeting Digest*, Baltimore, MD, p. 30, July 1996.

[5] J. Eric Bracken and Zoltan J. Cendes, “Transient analysis via electromagnetic fast–sweep methods and circuit models,” *13th Annual Review of Progress in Applied Computational Electromagnetics*, Monterey, CA, pp. 172–179, March 1997.

[6] Andreas C. Cangellaris and Li Zhao, “Reduced–order modeling of electromagnetic systems with Padé via Lanczos approximations,” *13th Annual Review of Progress in Applied Computational Electromagnetics*, Monterey, CA, pp. 148–155, March 1997.

[7] John R. Brauer and Gary C. Lizalek, “Microwave filter analysis using a new 3–D finite–element modal frequency method,” *IEEE Trans. Microwave Theory and Techniques*, v. 45, May 1997, pp. 810–818.

[8] John R. Brauer, ed., *What Every Engineer Should Know About Finite Element Analysis*, 2nd ed., New York, NY: Marcel Dekker, 1993.

[9] John Brauer and Avraham Frenkel, "S−parameters of microwave resonators computed by direct frequency and modal frequency finite element analysis," *13th Annual Review of Progress in Applied Computational Electromagnetics*, Monterey, CA, pp. 140−147, March 1997.

[10] P. Savi, D. Trinchero, R. Tascone, and R. Orta, "A new approach to the design of dual−mode rectangular waveguide filters with distributed coupling," *IEEE Trans. Microwave Theory and Techniques*, v. 45, Feb. 1997, pp. 221−228.

[11] A. E. Williams, "A four−cavity elliptic waveguide filter," *IEEE Trans. Microwave Theory and Techniques*, v. 18, Dec. 1970, pp. 1109−1113.

[12] Luciano Accatino, Giorgio Bertin, and Mauro Mongiardo, "A four−pole dual mode elliptic filter realized in circular cavity without screws," *IEEE Trans. Microwave Theory and Techniques*, v. 44, Dec. 1996, pp. 2680−2687.

[13] "CAD benchmark takes on filter design," *Microwave Engineering Europe*, Dec./Jan. 1997, pp. 7−12, and Feb./March 1997, pp. 9−10.

[14] Louis D. Komzsik, "Optimization of finite element software systems on supercomputers," in *Supercomputing in Engineering Analysis*, Hojjat Adeli, Ed. New York: Marcel Dekker, 1992, pp. 261−287.

[15] C. Lawson, R. Hanson, D. Kincaid, and F. Krogh, "Basic linear algebra subprograms for fortran usage," *ACM Trans. Math. Software*, v. 5, Oct. 1979, pp. 308−371.

[16] A. George and J. W. H. Liu, "The evolution of minimum degree ordering," *Tech. Report CS−87−06*, York University, U. K., 1987.

A Fast MEI Scheme for the Computation of Scattering by Very Large Cylinders

Y. Liu , K. Lan, K. K. Mei and E. K. N. Yung

Department of Electronic Engineering
City University of Hong Kong

Abstract: A fast scheme for measured equation of invariance (MEI) method is presented in this paper. The scheme combines a strategic technique of the interpolation and extrapolation of MEI coefficients with a special algorithm of cyclic block band matrix to fast solve the scattering problems of very large conducting cylinders. The circumferential dimension of scattering objects could exceed 10,000 wavelength. Computational speed could be 2-3 order faster than conventional MEI method. The fast scheme is especially applicable to scattering problems of very large conducting objects in which other numerical methods may fail.

1. Introduction

One of the advantages of the measured equation of invariance (MEI) method [1-4] is that the MEI can be used to solve scattering problems of normal size objects very fast. However when the target is very large, most numerical methods have difficulties in obtaining correct results. Even the MEI is no exception. In the case of the MEI, the reason is that the integration required to generate the MEI coefficients is very time consuming, and the results of the integration could be inaccurate which causes excessive errors in the resulting MEI coefficients and subsequent failure of the computation. In this paper, the interpolation and extrapolation technique for the MEI coefficients is used to overcome this problem. Such a technique allows us to extrapolate the MEI coefficients at several low frequencies to higher frequencies of interest. Thus the execution time for finding the MEI coefficients at the high frequencies can be greatly reduced, and the accuracy of the MEI coefficients can be greatly increased. The details of the interpolation and extrapolation technique for the

This work was supported in part by the City University of Hong Kong Strategic Grant 7000665 and Small Scale Research Grant 9030545.

MEI coefficients can be found in reference [5]. Compared with the total execution time for solving the large object scattering problems, after the utilization of the interpolation and extrapolation technique, finding the MEI coefficients becomes a minor part of the total computation time, and the solution of the MEI sparse matrix becomes the dominant part. In order to accelerate the whole computational process, the solution of the MEI matrix should be speeded up. To do so, one option is choosing parallel computation. Another is choosing special algorithms. For the parallel computation, a PVM (Parallel Virtual Machine) based parallel sparse matrix solver is investigated. A cluster of Pentium PC's is chosen as the hardware platform. Ethernet is used in connecting Pentiums as a network. When partition number of the MEI matrix is 4 - 16, a speedup of 7 for the MEI matrix can be achieved [6]. At the same time, a special algorithm, which is applicable for the structure of the MEI matrix, is developed. This special algorithm will be described in this paper. In 2-D scattering cases, the MEI matrix is such a sparse structure, in which all the elements are in a diagonal band area except a few off-diagonal elements at the beginning and the end of the mesh. This is a typical cyclic block tridiagonal structure. For such a special structure, we modify the Thomas algorithm [7], which directly solves block tridiagonal matrix, to meet our need. The computational time of our MEI matrix solver is proportional to N, the dimension of the matrix. For the scattering of a conducting circular cylinder with diameter 4,000 wavelength, the MEI solver is about 30 times faster than the Berkeley's sparse solver [8], and is about 4 times faster than the above PVM-based parallel sparse matrix algorithm. Thus using the fast MEI scheme in this paper, which combines the interpolation and extrapolation technique of the MEI coefficients with algorithm of the cyclic block band matrix, the scattering problems of very large cylinders can be solved much faster than with the conventional MEI or MoM algorithms.

2. The MEI Sparse Matrix

For the MEI computation mesh, suppose N_S be total grid points per one layer, and N_L be the total number of layers. Five points difference-difference formula is used for the nodes on the inner layers of the mesh,

$$\sum_{i=0}^{i=4} c_i \phi_i = 0 \tag{1}$$

where $\{c_i\}$ are the finite-difference coefficients, ϕ_i is the field at node i. For the grids on the truncated boundary, the MEI equation is used

$$\sum_{i=0}^{n-1} a_i \phi_i = 0 \tag{2}$$

where $\{a_i\}$ are the MEI coefficients to be determined, and $n-1$ is the number of neighbors of node 0.

After boundary conditions of the conducting object are applied, the MEI-FD matrix is nested in the following form,

$$\begin{bmatrix} B_1 & C_1 & 0 & 0 & \cdots & 0 & A_1 \\ A_2 & B_2 & C_2 & 0 & \cdots & 0 & 0 \\ 0 & A_3 & B_3 & C_3 & 0 & \cdots & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & \cdots & 0 \\ \vdots & \vdots & \cdots & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & \cdots & 0 & A_{N_s-1} & B_{N_s-1} & C_{N_s-1} \\ C_{N_s} & 0 & \cdots & 0 & 0 & A_{N_s} & B_{N_s} \end{bmatrix} \begin{bmatrix} \Phi_1 \\ \Phi_2 \\ \Phi_3 \\ \vdots \\ \vdots \\ \Phi_{N_s-1} \\ \Phi_{N_s} \end{bmatrix} = \begin{bmatrix} \Phi_1^I \\ \Phi_2^I \\ \Phi_3^I \\ \vdots \\ \vdots \\ \Phi_{N_s-1}^I \\ \Phi_{N_s}^I \end{bmatrix} \tag{3}$$

where $\{A_i\}$ $\{B_i\}$ and $\{C_i\}$ are $(N_L-1)\times(N_L-1)$ matrices, $\{\Phi_i\}$ and $\{\Phi_i^I\}$ are $(N_L-1)\times 1$ column vectors which represent the unknown scattered fields and known incident fields, respectively.

Obviously, the MEI matrix, i. e. eq. (3), is a typical cyclic block band matrix, in which all the elements are in a diagonal band area except a few off-diagonal elements at the beginning and the end of the mesh.

3. Strategy to obtain MEI coefficients

In this section, we will briefly describe how to use the interpolation and extrapolation technique to obtain MEI coefficients at the high frequencies of interest from those calculated at low frequencies being first calculated.

The MEI coefficients have the characteristics of spatial interpolability, frequency interpolability and frequency extrapolability [5] when the ratio of the wavelength and the discretization size stays the same. The MEI coefficients are interpolable between nodes, i.e., if the nodes are rearranged for different frequencies to keep the ratio invariant, we may obtain the MEI equations at the nodes of the new distribution from interpolations of the MEI coefficients at the nodes of the original distribution. Fig.1 shows the arrangements of the nodes at three different frequencies increased by a factor of two. The mesh size is reduced by a factor of two, at the same time, the spacing between the mesh boundary and object boundary is also reduced by a factor of two.

We expect the MEI coefficients to smoothly approach a limit when the frequency increases to the optical region. Our expectation is based on the similarity of the local geometry of a node as the mesh boundary approaches the object boundary and on the fact that in geometrical optics the same rules of calculation are applied to all frequencies. Numerical examinations in [5] have indicated that the MEI coefficients do approach a limit with the increase in frequency, and the accuracy of the MEI coefficients obtained by the extrapolation is enough for obtaining accurate results. We express the MEI coefficients at node 0 as polynomials of the normalized wavelength with the degree J-1,

$$a_i(\lambda) = \sum_{j=0}^{J-1} \alpha_{i,j} \lambda^j \tag{4}$$

Based on the mesh organization depicted in Fig.1, we calculate the MEI coefficients at few normalized measuring wavelength such as 2, 1, 0.5, 0.25, ... with number J, by using the conventional MEI method [1]. Then the coefficients $\{\alpha_{i,j}\}$ in eq. (4) can be found. Once the coefficients $\{\alpha_{i,j}\}$ are obtained, the MEI coefficients at any normalized wavelength can be determined by using eq.(4). Since we only calculate the MEI coefficients at the low frequencies through the integration, computational time is very small. In addition, the interpolation and extrapolation time for obtaining the MEI coefficients at the high frequency of interest is also small. Compared with direct calculation of the MEI coefficients at the high frequency by the integration, the computation time can be tens to thousands of times smaller depending on how large the scattering objects are.

The technique presented above is not just a way to obtain the MEI coefficients with economical computation time, but a way to obtain the MEI coefficients accurately for the very large electromagnetic boundary value problems.

4. The Fast Solver for the MEI Matrix

We have pointed out in section 3, that the CPU time can be greatly reduced if the interpolation and extrapolation technique is used in determining the MEI coefficients at the high frequency. Thus most of the CPU time will be used for solving the MEI matrix. In order to reduce the execution time even further, a fast MEI matrix solver has been developed.

As seen in section 2, most elements in the MEI matrix are tridiagonal nonzero block elements, and only few elements are off-tridiagonal nonzero block elements on each reverse-diagonal corner. This is a typical cyclic tridiagonal block structure. For such a special structure, the direct algorithm called Thomas algorithm [7], which has been used to get the solutions of tridiagonal block matrix systems, is expanded to solve this kind of cyclic tridiagonal block matrix systems. This is the MEI matrix solver. Unlike the conventional sparse matrix routine, the MEI matrix solver does not need to pivot and reorder the matrix before solving the matrix. All the MEI matrix solver does is solving the block matrix analytically. In followings, we will briefly describe this algorithm.

Using the first $N_S - 1$ equations in eq.(3), the forward substitution can be completed to obtain the coefficient matrices $\{\Delta_i\}, \{P_i\}$ and $\{Q_i\}$ recursively in the following formula,

$$\begin{bmatrix} P_1 & C_1 & 0 & \cdots & 0 \\ 0 & P_2 & C_2 & \cdots & 0 \\ 0 & 0 & \ddots & \ddots & 0 \\ 0 & \ddots & \ddots & P_{N_S-2} & C_{N_S-2} \\ 0 & 0 & \cdots & 0 & P_{N_S-1} \end{bmatrix} \begin{bmatrix} \Phi_1 \\ \Phi_2 \\ \vdots \\ \Phi_{Ns-2} \\ \Phi_{N_S-1} \end{bmatrix} = \begin{bmatrix} \Delta_1 - Q_1 \Phi_{N_S} \\ \Delta_2 - Q_2 \Phi_{N_S} \\ \vdots \\ \Delta_{N_S-2} - Q_{N_S-2} \Phi_{N_S} \\ \Delta_{N_S-1} - Q_{N_S-1} \Phi_{N_S} \end{bmatrix} \quad (5)$$

where $P_1 = B_1$, $\Delta_1 = \Phi_1^I$, $Q_1 = A_1$, and

$$P_i = B_i - A_i P_{i-1}^{-1} C_{i-1} \quad (6)$$

$$\Delta_i = \Phi^I{}_i - A_i P_{i-1}^{-1} \Delta_{i-1} \quad (7)$$

$$Q_i = -A_i P_{i-1}^{-1} Q_{i-1} \quad (8)$$

$i = 2,3,\ldots,N_s - 1$. And the unknown sub-matrix Φ_{N_S} can be determined by

$$\Delta_{N_S} = \Phi_{N_S}^I - A_{N_S} P_{N_S-1} \Delta_{N_S-1} - C_{N_S} W_1 \quad (9)$$

$$P_{Ns} = B_{N_S} - A_{N_S} P_{N_S-1} \left(C_{N_S-1} + Q_{N_s=1} \right) - C_{N_S} W_2 \quad (10)$$

$$\Phi_{N_S} = P_{N_S}^{-1} \Delta_{N_S} \quad (11)$$

where the matrices W_1 and W_2 can be determined during the above forward substitution recursively. Then, using backward substitution, all the unknown sub-matrices can be recursively calculated by,

$$\Phi_i = P_i^{-1}(\Delta_i - C_i\Phi_{i+1} - Q_i\Phi_{N_S}) \tag{12}$$

where $i = N_s - 1, \ldots, 1$.

In general, if a large number of mesh buffer layers is utilized for the finite difference, all the nonzero sub-matrices in the MEI matrix are still very sparse too. The operators between these sub-matrices can be carried out by general sparse matrix solvers to speed up the computation. However, for 2-D scattering problems, only 2 - 3 mesh layers are needed to obtain robust results in the MEI method [1]. In these cases, the number of zero elements in the sub-matrices is small. Thus the zero elements in the sub-matrices are directly treated as nonzero elements, and the additional matrix fill for these zero elements is small. For example, for three layer mesh with a 6 node MEI's sub-mesh scheme, the total number of nonzero elements is $11N_S$, while the total elements to be filled in eq.(3) is $14N_S$. Another advantage is that since the size of the sub-matrices is small, all the operators between the sub-matrices can be given analytically, thus the calculation for the zero elements is avoided.

The CPU time of the MEI matrix solver is nearly proportional to N_S, the number of nodes per one layer, rather than proportional to N_S^{α} $(\alpha > 1)$ as is the case with the general sparse matrix solver [5]. As the dimension of the MEI matrix increases, the time-saving of the MEI matrix solver will increase.

5. Statistics of CPU Time

The CPU time comparison of the MEI matrix solver with the Berkeley's sparse package [8] is presented in Fig.2, which shows that when N_S is small, the CPU time of the MEI matrix solver and the Berkeley's sparse solver are almost identical. However, when the size of the MEI matrix increases, the CPU time of the MEI matrix solver is much less than the CPU time of the Berkeley's sparse solver. For solving the scattering of a cylinder at above 10^5 wavelength circumferential dimension, the MEI matrix solver is about 30 times faster than the Berkeley's general sparse solver. The CPU time for obtaining MEI coefficients using the interpolation and extrapolation technique is also plotted in Fig.2. It is found that without using the interpolation and extrapolation technique, the time consumed by the direct integration to obtain MEI coefficients for a geometry at 2,000 wavelength circumferential dimension will exceed 10hrs, but the interpolation and extrapolation technique needs a few minutes only.

6. Numerical Results

The above scheme allows us to fast solve the scattering problems of large 2-D conducting convex objects. For illustration purpose, we first calculate TM scattering surface current density of a conducting circular cylinder with diameter d = 4,000λ (wavelength) under 0^0 plane wave incidence. The surface current densities compared to the result of geometrical optics are shown in Fig.3. To solve this problem, the MEI coefficients at the lower frequencies are calculated at normalized wavelength 0.8, 0.4, 0.2, 0.1 for a cylinder of diameter d = 10λ. The normalized wavelength of the high frequency is 0.0025. The CPU time to find the MEI coefficients is about 2 minutes. The CPU time to solve the MEI matrix solver is about 124 seconds. But the Berkeley's sparse solver should take the CPU time of 72 minutes.

The second example is the scattering of a square conducting cylinder of 64 meter in circumerferial dimension. The normalized wavelength 2, 1, 0.5, 0.25 of the low frequencies is extrapolated to the wavelength of interest 0.0625, for which the ratio of the circumerferial dimension to the wavelength is 1,024. Fig.4 gives the surface current densities by a 30^0 TE incidence wave illumination. Fig.5 shows the remarkable agreement between the fast MEI scheme and the GTD on the RCS when an incidence wave with angle 45^0 illuminates with the same configuration of Fig.4.

7. Conclusion

In this paper, we present a fast numerical scheme for the MEI method to solve the scattering problems of the very large conducting cylinders in which circumferential dimension exceeds 10,000 wavelength, in a common workstation. Since the scheme combines the extrapolation and interpolation technique of the MEI coefficients with the special algorithm of cyclic block band matrix, the whole computation can be speeded up to 2-3 orders with reasonable numerical accuracy.

Reference

[1] K. K. Mei, R. Pous, Z. Q. Chen and Y. W. Liu, "The Measured Equation of Invariance: A New Concept in Field Computation," IEEE Trans. on Antenna and Propagat., Vol. 42, No. 3, pp. 320~327, 1994

[2] M. Prouty, S. Schwarz, K. K. Mei and Y. Liu, "A New Approach to Quasistatic Analysis with Application to Microstrip," IEEE Microwave and Guided Wave Letters, Vol.3, No.9, pp.302, 1993

[3] J. M. Rius, "Integral Formulation of the Measured Equation of Invariance," Electronic Letters, Vo.32, No.1,pp.23~25,1996

[4] J. Jevtic and R. Lee, "A Theoretical and Numerical Analysis of the Measured Equation of Invariance," IEEE Trans. on Antenna and Propagat., Vol.42, No.8, pp.1097~1105, 1994.

[5] Y. Liu, K. N. Yung and K. K. Mei, "Interpolation, Extrapolation and Application of the Measured Equation of Invariance to Scattering by Very Large Cylinders," IEEE Trans. on Antenna and Propagat., Vol.45, No.9, pp.1325~1331, 1997

[6] R. M. M. Chen, G. F. Niu, Y. W. Liu, and K. K. Mei, "A PVM parallel sparse matrix equation solver to speed up computation of MEI method," IEEE Antenna and Propagat., Society International Symposium, Vol. 2, pp. 660 -663, Montreal, Canada, July 13-18, 1997.

[7] G. V. Davis, Numerical Methods in Engineering & Science, Allen & Unwin Ltd., 1986.

[8] K. S. Kundert, A. Sangiovanni-Vincentelli, Sparse Users' Guide - A Sparse Linear Equation Solver, University of California, Berkeley, 1988.

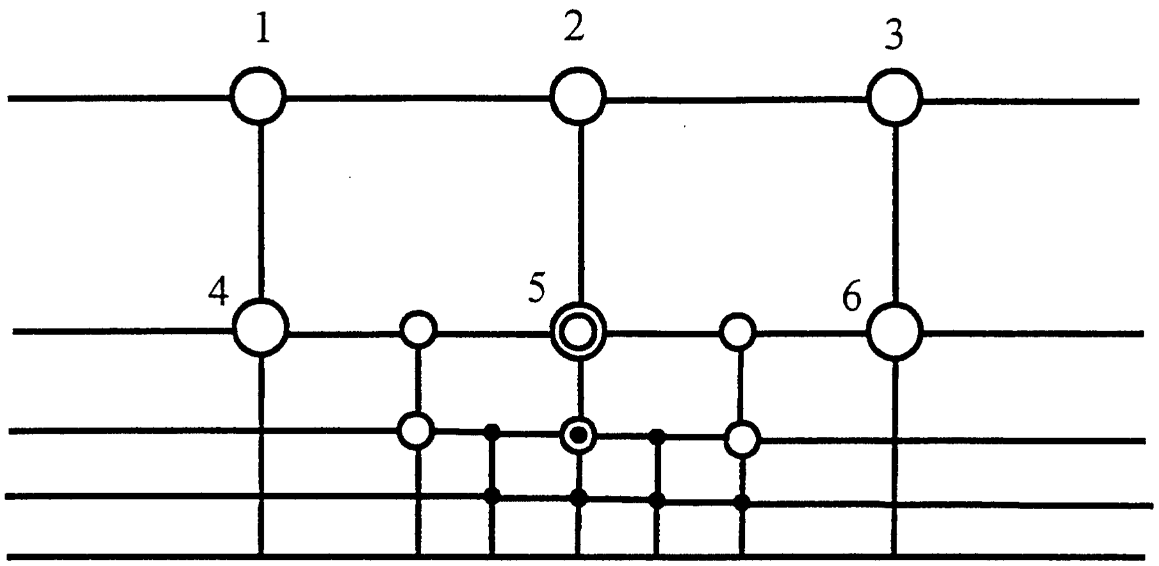


Fig. 1 Mesh strategy with the mesh size reduced by a factor of two.

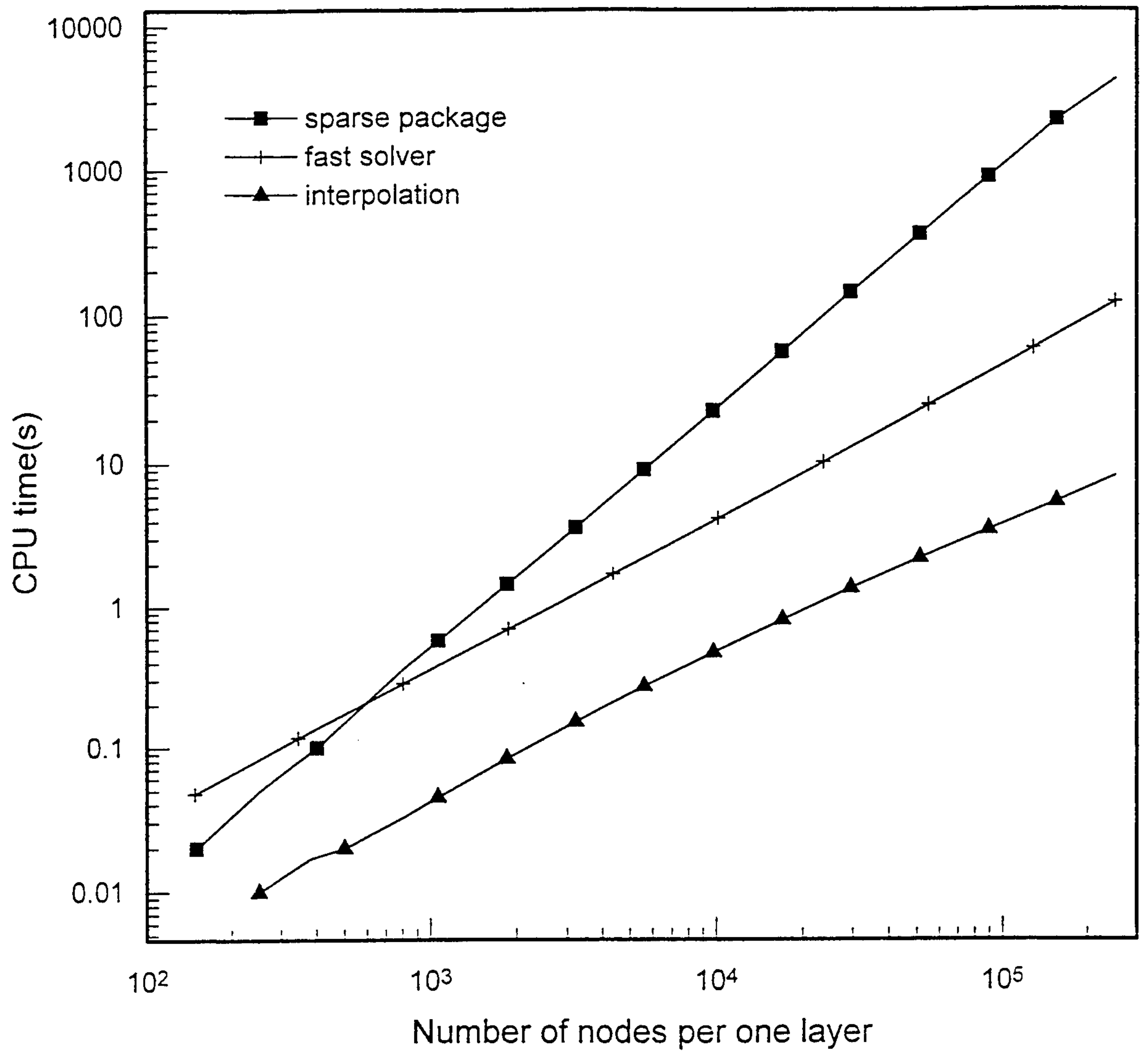


Fig. 2 CPU time vs. number of nodes per one layer on a Sun-Workstation.

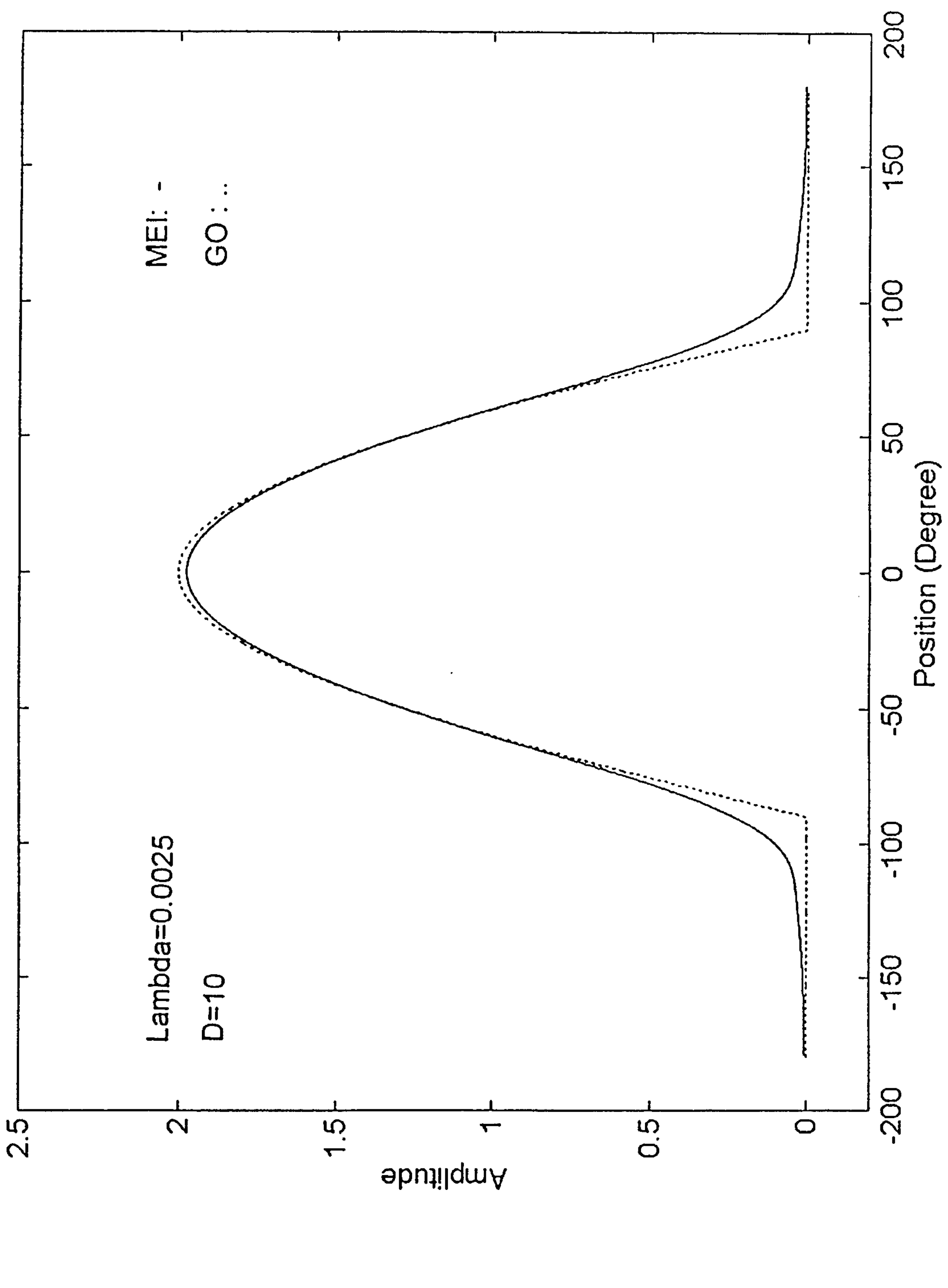


Fig. 3 Surface current distribution on a circular cylinder illuminated by a TM wave with d = 4,000λ.

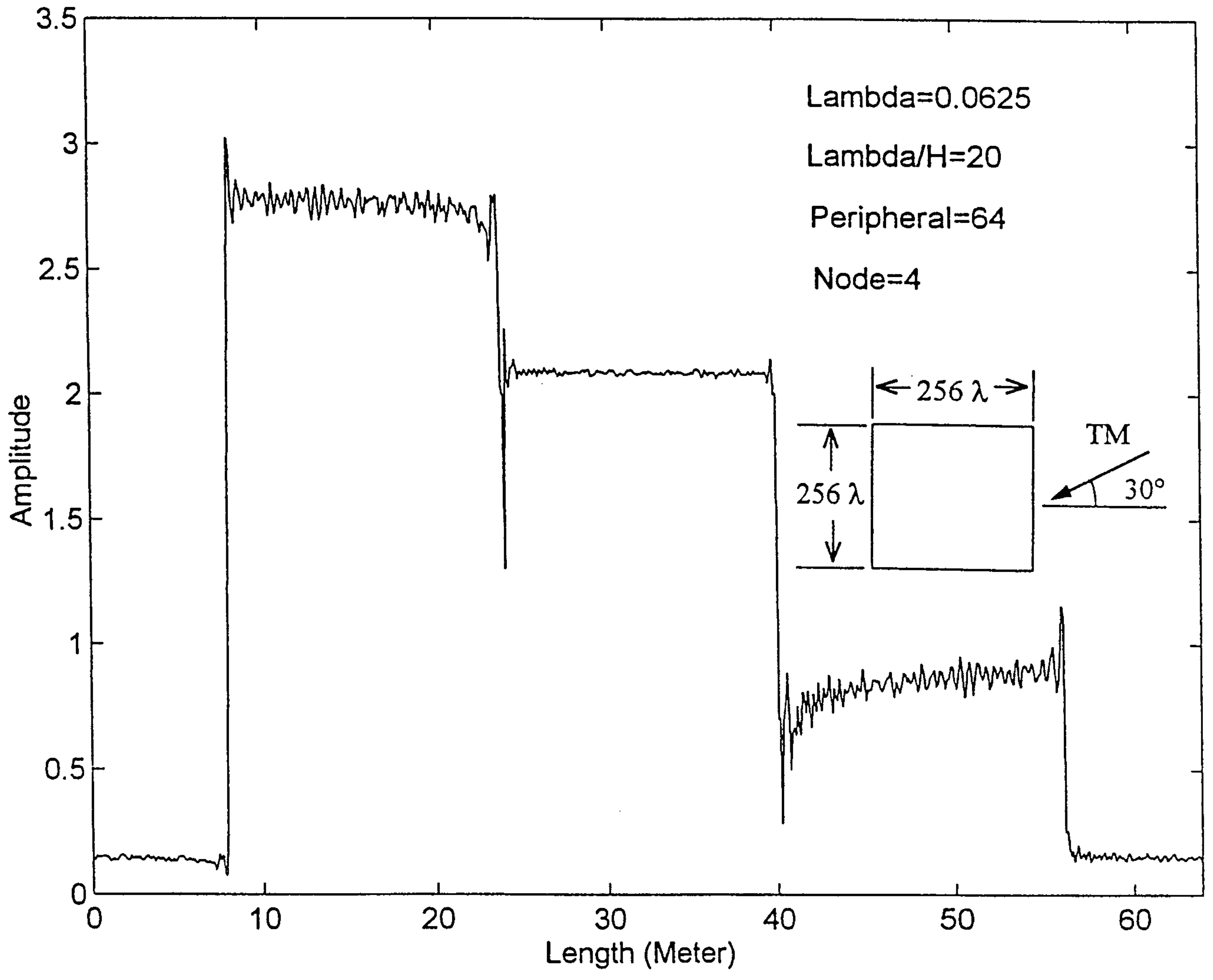


Fig. 4 Surface current distribution on a 256λ x 256λ square cylinder illuminated by a 30° TE wave incidence.

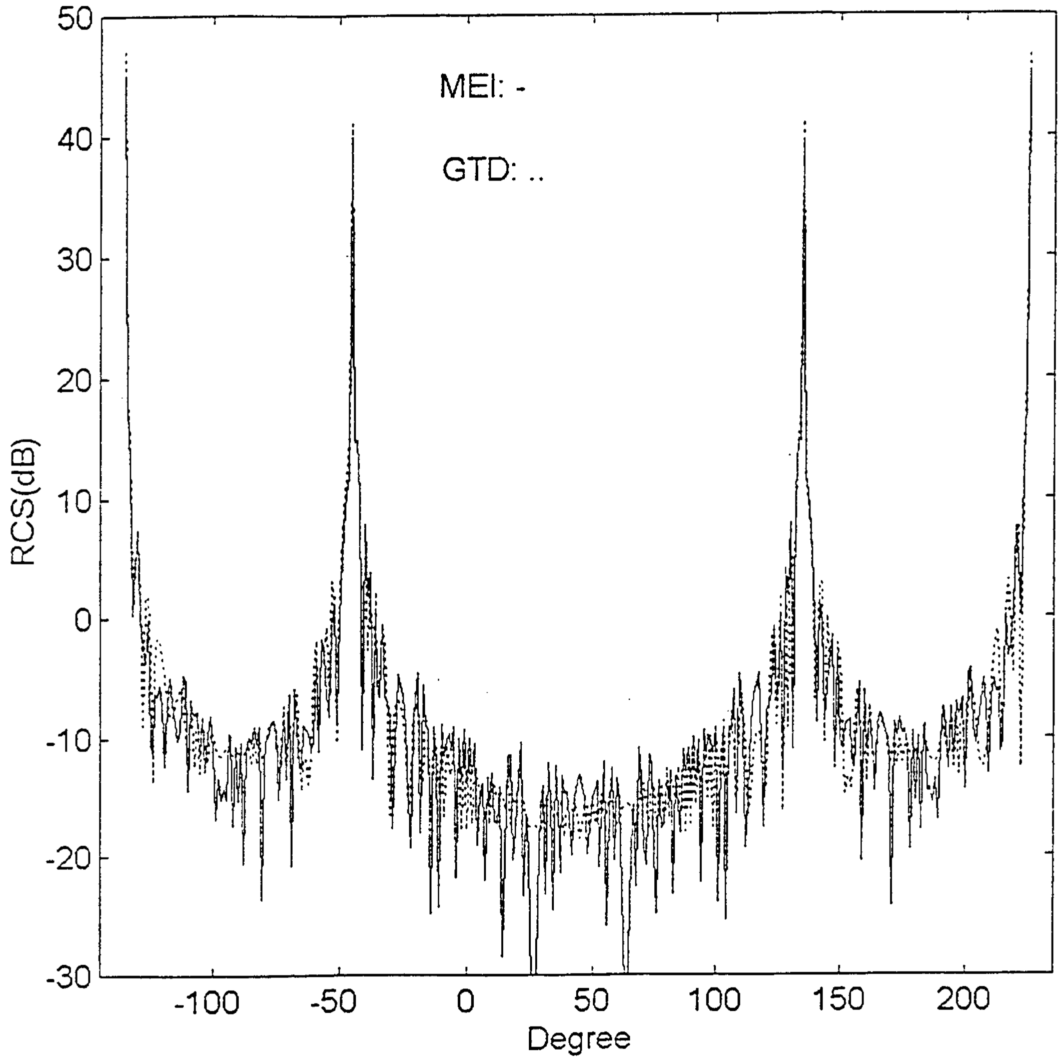


Fig. 5 Comparison of bistatic RCS of the scattering configuration in Fig. 4 illuminated by 45° TE wave incidence.

1998 INSTITUTIONAL MEMBERS

ANDREW CORPORATION
10500 W. 153rd Street
Orland Park, IL 60462

BOEING NORTH AMERICAN SVCS
1745 Jefferson Davis Hwy
Arlington, VA 22202

BRITISH AEROSPACE
FPC 267 PO Box 5
Filton, BRISTOL, UK BS12 7QW

BRITISH BROADCASTING CO R&D
Kingswood Warren
Tadworth, SURREY, UK KT20 6NP

CAMBRIDGE CONSULTANTS, LTD.
Milton Road, Science Park
Cambridge, CAMBS, UK CB5 4DW

COMMUNICATOR CEC AB
Box 1310
Solna, SWEDEN 17125

CSIRO, CTR FOR ADV. TECH.
PO Box 883
Kenmore, QLD, AUSTRALIA 4069

CULHAM LAB
UK Atomic Energy Authority
Abingdon, OXFORD, UK OX14 3DB

DARTMOUTH COLLEGE
Feldberg Library, 6193 Murdough Ctr.
Hanover, NH 03755-3560

DCN INTERNATIONAL
153 04 Rockaway Blvd.
Jamaica, NY 11434-5410

DEFENCE TECH & PROCUREMENT
NEMP Lab, AC-Zentrum
Spiez, SWITZERLAND CH 3700

DEFENSE RESEARCH ESTAB. LIB.
3701 Carling Avenue
Ottowa, ON, CANADA K1A 0Z4

DERA
Common Road, Funtington
Chichester, UK PO18 9PD

DEUTSCHE TELEKOM AG
PO Box 10-00-03
Darmstadt, GERMANY D-64 276

DSTO LIBRARY
Box 1500
Salisbury, SA, AUSTRALIA 5108

E.T.S.E. TELECOMUNICACIONES
Campus Lagoas Marcosende
Vigo, SPAIN 36200

ELECTRONICS RESEARCH INC.
7777 Gardner Road
Chandler, IN 47610-9219

ERA TECHNOLOGY LTD.
Cleve Road
Leatherhead, SURREY, UK KT22 7SA

ERICSSON SAAB AVIONICS AB
Electromagnetic Technology
Linkoeping, SWEDEN SE 58188

ESCUELA TRANSMISS. EJERCITO
Ctr. Extremadura, KM, 10,200
Madrid, SPAIN 28024

FANFIELD LTD.
Braxted Park
Witham, ESSEX, UK CM8 3XB

FGAN/FHP/AUS
Neuenahrer Strasse 20
Wachtberg, Werth, GERMANY 53343

FIRSTMARK TECHNOLOGIES
85 St. Charles Quest St.
Longueuil, PQ, CANADA J4H 1C5

GEC MARCONI RES. CTR. LIB.
W. Hanningfield Road, Gt. Baddow
Chelmsford, ESSEX, UK CM2 8HN

HARRIS CORPORATION
1680 University Avenue
Rochester, NY 14610-9983

HKUST, UNIVERSITY LIBRARY
Clear Water Bay Road
Kowloon, HONG KONG

HUGHES RESEARCH LIBRARY
3011 Malibu Canyon Road
Malibu, CA 90265-4737

HUNTING ENGINEERING LTD.
Reddings Wood, Ampthill
Bedford, UK MK45 2HD

HYDRO-QUEBEC INST. OF RES.
1800 Boul. Lionel-Boulet
Varennes, PQ, CANADA J3X 1S1

IABG MBH, TRM
Einsteinstrasse 20
Ottobrunn, GERMANY D 85521

IIT RESEARCH INSTITUTE
185 Admiral Cochrane Drive
Annapolis, MD 21401-7396

IMAGINEERING LTD.
95 Barber Greene Road, Suite 112
Toronto, ON, CANADA M3C 3E9

INFORMATION CENTRE
A4 Bldg. Ively Road
Farnsborough, HAMPS. UK GU14 0LK

IPS RADIO & SPACE SVC/LIBRARY
PO Box 5606
W. Chatswood, AUSTRALIA 2057

LICOSA LIBRARY
Via Duca Di Calabria 1/1
Florence, ITALY 50125

LINDA HALL LIBRARY
5109 Cherry Street
Kansas City, MO 64110-2498

METAWAVE COMMUNICATIONS
8700 148th Avenue, N.E.
Redmond, WA 98052

MISSISSIPPI STATE UNIV. LIBRARY
PO Box 9570
Mississippi State, MS 39762

MITRE CORPORATION LIBRARY
202 Burlington Road
Bedford, MA 01730-1407

MOD(PE) DGSS
Abbey Wood #54, PO Box 702
Bristol, UK BS12 7DU

MOTOROLA
2001 N. Division Street
Harvard, IL 60033

MPB TECHNOLOGIES INC.
151 Hymus Blvd.
Pointe-Claire, PQ, CANADA H9R 1E9

NATIONAL AEROSPACE LAB, NLR
Anthony Fokkerwek 2
Emmeloord, NETHERLANDS 8300

NATL RADIOLOGICAL PROT. BD.
Chilton
Didcot, OXON, UK OX11 0RG

NAVAL RESEARCH LABORATORY
C. Office
Washington, DC 20375

NGIC
220 7th Street, NE
Charlottesville, VA 22902-5396

NIKSAR
35/45 Gilbey Road
Mt. Waverley, VIC, AUSTRALIA 3149

NNR AIR CARGO SERVICE
Hook Creed Blvd. & 145th Avenue
Valley Stream, NY 11581

NORTEL TECHNOLOGY
London Road
Harlow, ESSEX, UK CM17 9NA

NTT WIRELESS SYSTEMS LABS.
1-1 Hikarinooka Yokosuka-Shi
Kanagawa, JAPAN 239-0847

PENN STATE UNIVERSITY LIBRARY
Pattee Library
University Park, PA 16802

PHILIPS RESEARCH LAB LIBRARY
Cross Oak Lane, Salfords
Redhill, SURREY. UKRH1 5HA

PHILIPS RESEARCH
Prof. Holstlaan 4
Eindhoven, NETHERLANDS 11172J

QUEEN MARY & WESTFIELD COLL.
Mile End Road
London, UK E1 4NS

RAND AFRIKAANS UNIVERSITY
PO Box 524, Aucklandpark
Johannesburg, S AFRICA 2006

RAYTHEON E-SYSTEMS
PO Box 6056
Greenville, TX 75403

ROCKWELL INTERNATIONAL
350 Collins Road
Cedar Rapids, IA 52498

SONY CORPORATION
174 Fujitsukacho, Hodogaya Ku
Yokohama MZ, JAPAN 240

SOUTHWEST RESEARCH INST.
6220 Culebra Road
San Antonio, TX 78238

SWETS SUBSCRIPTION SERVICE
440 Creamery Way, Suite A
Exton, PA 19341

TAMPERE UNIVERSITY OF TECH
PO Box 692
Tampere, FINLAND 33101

TASC - LIBRARY
55 Walkers Brook Drive
Reading, MA 01867-3297

TECHNISCHE UNIV. DELFT
Mekelweg 4, Delft
HOLLAND,NETHERLANDS 2628 CD

TELEBRAS - CPQD, LIB.
Rod. Campinas
Campinas, SP BRAZIL 13088-061

TELSTRA INFO & LIBRARY SVC.
Private Bag 37
Clayton, VIC, AUSTRALIA 3168

TELSTRA RESEARCH LABS
770 Blackburn Road
Clayton, VIC, AUSTRALIA 3168

UNIV OF CENTRAL FLORIDA LIB.
PO Box 162440
Orlando, FL 32816-2440

UNIV OF COLORADO LIBRARY
Campus Box 184
Boulder, CO 80309-0184

UNIV OF MISSOURI-ROLLA LIB.
1870 Miner Circle
Rolla, MO 65409-0001

UNIVERSITAT DER BUNDESWEHR
Werner Heisenberg Weg 39
Neubiberg, GERMANY D-85577

US COAST GUARD ACADEMY
15 Mohegan Avenue
New London, CT 06320-4195

VECTOR FIELDS LTD.
24 Bankside Kidlington
Oxford, UK OX5 1JE

VIT, TECHNICAL RESEARCH. CTR.
PO Box 1202
Espoo, FINLAND FIN-02044

THE APPLIED COMPUTATIONAL ELECTROMAGNETICS SOCIETY

ACES COPYRIGHT FORM

This form is intended for original, previously unpublished manuscripts submitted to ACES periodicals and conference publications. The signed form, appropriately completed, MUST ACCOMPANY any paper in order to be published by ACES. PLEASE READ REVERSE SIDE OF THIS FORM FOR FURTHER DETAILS.

TITLE OF PAPER:

AUTHORS(S)

PUBLICATION TITLE/DATE:

RETURN FORM TO:
Dr. Richard W. Adler
Naval Postgraduate School
Code EC/AB
833 Dyer Road, Room 437
Monterey, CA 93943-5121 USA

PART A - COPYRIGHT TRANSFER FORM

(NOTE: Company or other forms may not be substituted for this form. U.S. Government employees whose work is not subject to copyright may so certify by signing Part B below. Authors whose work is subject to Crown Copyright may sign Part C overleaf).

The undersigned, desiring to publish the above paper in a publication of ACES, hereby transfer their copyrights in the above paper to The Applied Computational Electromagnetics Society (ACES). The undersigned hereby represents and warrants that the paper is original and that he/she is the author of the paper or otherwise has the power and authority to make and execute this assignment.

Returned Rights: In return for these rights, ACES hereby grants to the above authors, and the employers for whom the work was performed, royalty-free permission to:

1. Retain all proprietary rights other than copyright, such as patent rights.
2. Reuse all or portions of the above paper in other works.
3. Reproduce, or have reproduced, the above paper for the author's personal use or for internal company use provided that (a) the source and ACES copyright are indicated, (b) the copies are not used in a way that implies ACES endorsement of a product or service of an employer, and (c) the copies per se are not offered for sale.
4. Make limited distribution of all or portions of the above paper prior to publication.
5. In the case of work performed under U.S. Government contract, ACES grants the U.S. Government royalty-free permission to reproduce all or portions of the above paper, and to authorize others to do so, for U.S. Government purposes only.

ACES Obligations: In exercising its rights under copyright, ACES will make all reasonable efforts to act in the interests of the authors and employers as well as in its own interest. In particular, ACES REQUIRES that:

1. The consent of the first-named author be sought as a condition in granting re-publication permission to others.
2. The consent of the undersigned employer be obtained as a condition in granting permission to others to reuse all or portions of the paper for promotion or marketing purposes.

In the event the above paper is not accepted and published by ACES or is withdrawn by the author(s) before acceptance by ACES, this agreement becomes null and void.

AUTHORIZED SIGNATURE — TITLE (IF NOT AUTHOR)

EMPLOYER FOR WHOM WORK WAS PERFORMED — DATE FORM SIGNED

PART B - U.S. GOVERNMENT EMPLOYEE CERTIFICATION

(NOTE: If your work was performed under Government contract but you are not a Government employee, sign transfer form above and see item 5 under Returned Rights).

This certifies that all authors of the above paper are employees of the U.S. Government and performed this work as part of their employment and that the paper is therefore not subject to U.S. copyright protection.

AUTHORIZED SIGNATURE — TITLE (IF NOT AUTHOR)

NAME OF GOVERNMENT ORGANIZATION — DATE FORM SIGNED

PART C - CROWN COPYRIGHT

(Note: ACES recognizes and will honor Crown Copyright as it does U.S. Copyright. It is understood that, in asserting Crown Copyright, ACES in no way diminishes its rights as publisher. Sign only if *ALL* authors are subject to Crown Copyright.

This certifies that all authors of the above Paper are subject to Crown Copyright. (Appropriate documentation and instructions regarding form of Crown Copyright notice may be attached).

AUTHORIZED SIGNATURE TITLE OF SIGNEE

NAME OF GOVERNMENT BRANCH DATE FORM SIGNED

Information to Authors

ACES POLICY

ACES distributes its technical publications throughout the world, and it may be necessary to translate and abstract its publications, and articles contained therein, for inclusion in various compendiums and similar publications, etc. When an article is submitted for publication by ACES, acceptance of the article implies that ACES has the rights to do all of the things it normally does with such an article.

In connection with its publishing activities, it is the policy of ACES to own the copyrights in its technical publications, and to the contributions contained therein, in order to protect the interests of ACES, its authors and their employers, and at the same time to facilitate the appropriate re-use of this material by others.

The new United States copyright law requires that the transfer of copyrights in each contribution from the author to ACES be confirmed in writing. It is therefore necessary that you execute either Part A-Copyright Transfer Form or Part B-U.S. Government Employee Certification or Part C-Crown Copyright on this sheet and return it to the Managing Editor (or person who supplied this sheet) as promptly as possible.

CLEARANCE OF PAPERS

ACES must of necessity assume that materials presented at its meetings or submitted to its publications is properly available for general dissemination to the audiences these activities are organized to serve. It is the responsibility of the authors, not ACES, to determine whether disclosure of their material requires the prior consent of other parties and if so, to obtain it. Furthermore, ACES must assume that, if an author uses within his/her article previously published and/or copyrighted material that permission has been obtained for such use and that any required credit lines, copyright notices, etc. are duly noted.

AUTHOR/COMPANY RIGHTS

If you are employed and you prepared your paper as a part of your job, the rights to your paper initially rest with your employer. In that case, when you sign the copyright form, we assume you are authorized to do so by your employer and that your employer has consented to all of the terms and conditions of this form. If not, it should be signed by someone so authorized.

NOTE RE RETURNED RIGHTS: Just as ACES now requires a signed copyright transfer form in order to do "business as usual", it is the intent of this form to return rights to the author and employer so that they too may do "business as usual". If further clarification is required, please contact: The Managing Editor, R.W. Adler, Naval Postgraduate School, Code EC/AB, Monterey, CA, 93943, USA (408)656-2352.

Please note that, although authors are permitted to re-use all or portions of their ACES copyrighted material in other works, this does not include granting third party requests for reprinting, republishing, or other types of re-use.

JOINT AUTHORSHIP

For jointly authored papers, only one signature is required, but we assume all authors have been advised and have consented to the terms of this form.

U.S. GOVERNMENT EMPLOYEES

Authors who are U.S. Government employees are not required to sign the Copyright Transfer Form (Part A), but any co-authors outside the Government are.

Part B of the form is to be used instead of Part A only if all authors are U.S. Government employees and prepared the paper as part of their job.

NOTE RE GOVERNMENT CONTRACT WORK: Authors whose work was performed under a U.S Government contract but who are not Government employees are required to sign Part A-Copyright Transfer Form. However, item 5 of the form returns reproduction rights to the U.S. Government when required, even though ACES copyright policy is in effect with respect to the reuse of material by the general public.

THE APPLIED COMPUTATIONAL ELECTROMAGNETICS SOCIETY

CALL FOR PAPERS

The 15th Annual Review of Progress in Applied Computational Electromagnetics

March 15-19, 1999

Naval Postgraduate School, Monterey, California

"Share Your Knowledge and Expertise with Your Colleagues"

The Annual ACES Symposium is an ideal opportunity to participate in a large gathering of EM analysis enthusiasts. The purpose of the Symposium is to bring analysts together to share information and experience about the practical application of EM analysis using computational methods. The symposium offerings include technical presentations, demonstrations, vendor booths and short courses. All aspects of electromagnetic computational analysis are represented. Contact for details.

Technical Program Chairman
Randy Haupt
EE Dept., 260
University of Nevada
Reno, NV 89557-0153
Phone: (702)-784-6927
Fax: (702)-784-6627
Email:haupt@ee.unr.edu

Symposium Administrator
Richard W. Adler
ECE Dept/Code EC/AB
Naval Postgraduate School
833 Dyer Road, Room 437
Monterey, CA 93943-5121
Phone: (408) 646-1111
Fax: (408) 649-0300
Email:rwa@ibm.net

Symposium Co-Chairman
Indira Chatterjee
EE Dept., 260
University of Nevada
Reno, NV 89557-0153
Phone: (702)-784-1346
Fax: (702)-784-6627
Email:indira@ee.unr.edu

Symposium Co-Chairman
James Hensen
EE Dept., 260
University of Nevada
Reno, NV 89557-0153
Phone: (702)-784-6929
Fax: (702)-784-6627
Email:jmb@proton.ee.unr.edu

The ACES Symposium is a highly influential outlet for promoting awareness of recent technical contributions to the advancement of computational electromagnetics. Attendance and professional program paper participation from non-ACES members and from outside North America are encouraged and welcome.

Early Registration Fees; (approximate*)	**ACES MEMBERS**	**$255**
	NON-MEMBER	**$295**
	STUDENT/RETIRED/UMEMPLOYED	**$115 (no proceedings)**
	STUDENT/RETIRED/UNEMPLOYED	**$150 (includes proceedings)**

*The exact fee will be announced later. Each conference registration is entitled to publish two papers in the proceedings free of charge. Excess pages over a paper limit of 8 will be charged $15/page.

1999 ACES Symposium

Sponsored by:	ACES, NPS, DOE/LLNL, U of NV, BYU, DOD, SIAM, NCCOSC and AMTA
in cooperation with:	The IEEE Antennas and Propagation Society, the IEEE Electromagnetic Compatibility Society and USNC/URSI

Visit ACES on line at: www.emclab.umr.edu/aces

THE APPLIED COMPUTATIONAL ELECTROMAGNETICS SOCIETY

CALL FOR PAPERS

The 15th Annual Review of Progress in Applied Computational Electromagnetics

Papers may address general issues in applied computational electromagnetics, or may focus on specific applications, techniques, codes, or computational issues of potential interest to the Applied Computational Electromagnetics Society membership. Area and topics include:

- Code validation
- Code performance analysis
- Computational studies of basic physics
- Examples of practical code application
- New codes, algorithms, code enhancements, and code fixes
- Computer Hardware Issues
- Partial list of applications:

Partial list of applications:	antennas	wave propagation
	radar imaging	radar cross section
	shielding	bioelectromagnetics
	EMP, EMI/EMC	visualization
	dielectric & magnetic materials	inverse scattering
	microwave components	MIMIC technology
	fiberoptics	remote sensing & geophysics
	communications systems	propagation through plasmas
	eddy currents	non-destructive evaluation

- Partial list of techniques:

Partial list of techniques:	frequency-domain & time-domain techniques	
	integral equation & differential equation techniques	
	finite difference & finite element analysis	
	diffraction theories	physical optics
	modal expansions	perturbation methods
	hybrid methods	moment methods

INSTRUCTIONS AND TIMETABLE FOR AUTHORS

November 20, 1998: Submission deadline. Submit four copies of a full-length, camera-ready paper to the Technical Program Chairman. Please supply the following data for the corresponding author: name, address, email address, FAX, and phone numbers. See below for instructions for the format of paper.

December 21, 1998: Authors notified of acceptance.

PAPER FORMATTING REQUIREMENTS

The recommended paper length is 6 pages, with 8 pages as a maximum, including figures. The paper should be camera-ready (good resolution, clearly readable when reduced to the final print of 6 x 9 inch paper). The paper should be printed on 8-1/2 x 11 inch papers with 13/16 side margins, 1-1/16 inch top margin, and 1 inch on the bottom. On the first page, place title 1-1/2 inches from top with author and affiliation beneath the title. Single spaced type using 10 or 12 point front size, entire text should be justified (flush left and flush right). No typed page numbers, but number your pages lightly in pencil on the back of each page.

SHORT COURSES

Short courses will be offered in conjunction with the Symposium covering numerical techniques, computational methods, surveys of EM analysis and code usage instruction. It is anticipated that short courses will be conducted principally on Monday March 15 and Friday March 19. Fees for **Half-day course** will be: $90 per person if booked before 1 March 99; $100, if booked from 1 March to 15 March 99; and $110 if booked at Conference time. **Full-day Courses** will be: $140 if booked before 1 March 1999; $150 if booked from 1 March to 15 March; $160 if booked at Conference time. **Short Course Attendance is not covered by the Symposium Registration Fee!**

EXHIBITS

Vendor booths and demonstrations will feature commercial products, computer hardware and software demonstrations, and small company capabilities.

ACES MEMBERSHIP - NEWSLETTER & JOURNAL SUBSCRIPTION FORM

please print

LAST NAME FIRST NAME MIDDLE INITIAL

COMPANY/ORGANIZATION/UNIVERSITY DEPARTMENT/MAIL STATION

PLEASE LIST THE ADDRESS YOU WANT USED FOR PUBLICATIONS

MAILING ADDRESS

CITY PROVINCE/STATE COUNTRY ZIP/POSTAL CODE

TELEPHONE FAX AMATEUR RADIO CALL SIGN

E-MAIL ADDRESS E-MAIL ADDRESS CAN BE INCLUDED IN ACES DATABASE ☐ YES ☐ NO

PERMISSION IS GRANTED TO HAVE MY NAME PLACED ON MAILING LISTS WHICH MAY BE SOLD ☐ YES ☐ NO

CURRENT SUBSCRIPTION PRICES

AREA	INDIVIDUAL SURFACE MAIL	INDIVIDUAL AIRMAIL	ORGANIZATIONAL (AIRMAIL ONLY)
U.S. & CANADA	( ) $ 65	( ) $ 65	( ) $115
MEXICO, CENTRAL & SOUTH AMERICA	( ) $ 68	( ) $ 70	( ) $115
EUROPE, FORMER USSR TURKEY, SCANDINAVIA	( ) $ 68	( ) $ 78	( ) $115
ASIA, AFRICA, MIDDLE EAST & PACIFIC RIM	( ) $ 68	( ) $ 85	( ) $115

FULL-TIME STUDENT/RETIRED/UNEMPLOYED RATE IS $25 FOR ALL COUNTRIES

CREDIT CARD USERS

IF YOU ARE PAYING BY CREDIT CARD & CARD *IS* YOUR OWN, YOU MUST, (1) PRINT AND SIGN YOUR NAME BELOW; (2) MAKE SURE YOUR COMPLETE ADDRESS IS LISTED ABOVE. IF THE CARD YOU ARE USING IS *NOT* YOUR CARD, THE CARD HOLDER MUST, (3) PRINT AND SIGN HIS/HER NAME AND, (4) ENTER HIS/HER COMPLETE ADDRESS BELOW.

PRINT FIRST AND LAST NAME OF CARD HOLDER SIGNATURE OF CARD HOLDER

MAILING ADDRESS

MAILING ADDRESS (cont)

METHOD OF PAYMENT
- ☐ A bank check for the total amount is enclosed.(1)
- ☐ Traveler's checks for the total amount are enclosed.(2)
- ☐ International Money Order is enclosed.(3)
- ☐ Charge to: ☐ MasterCard ☐ Visa. ☐ Discover ☐ Amex.(4)

Card No. | | | | | | | | | | | | | | | | |

Card Exp. Date:
Mo. _______ **Year** _____

MAKE CHECKS PAYABLE TO "ACES" and send to: RICHARD W. ADLER, EXEC. OFFICER, NAVAL POSTGRADUATE SCHOOL, ECE DEPT., CODE EC/AB, 833 DYER ROAD, ROOM 437, MONTEREY, CA 93943-5121

Non-USA participants may remit via (1) **Bank Checks**, if (a) drawn on a U.S. Bank, (b) have bank address, (c) contain series of (9) digit mandatory routing numbers; (2) **Traveler's Checks** (in U.S. $$); **(3) International Money Order** drawn in U.S. funds, payable in U.S.; (4) **Credit Cards:** Visa, Master Card, Discover Card, Amex.

Total Remittance (U.S. Dollars Only) $ _____________

July 1998

ADVERTISING RATES

	FEE	PRINTED SIZE
Full page	$200.	7.5" x 10.0"
1/2 page	$100.	7.5" x 4.7" or 3.5" x 10.0"
1/4 page	$ 50	3.5" x 4.7"

All ads must be camera ready copy.

Ad deadlines are same as Newsletter copy deadlines.

Place ads with Ray Perez, Newsletter Editor, Martin Marietta Astronautics, MS 58700, PO Box 179, Denver, CO 80201, USA. The editor reserves the right to reject ads.

ACES NEWSLETTER AND JOURNAL COPY INFORMATION

Issue	Copy Deadline
March	January 13
July	May 25
November	September 25

APPLIED COMPUTATIONAL ELECTROMAGNETICS SOCIETY JOURNAL

INFORMATION FOR AUTHORS

PUBLICATION CRITERIA

Each paper is required to manifest some relation to applied computational electromagnetics. **Papers may address general issues in applied computational electromagnetics, or they may focus on specific applications, techniques, codes, or computational issues.** While the following list is not exhaustive, each paper will generally relate to at least one of these areas:

1. Code validation. This is done using internal checks or experimental, analytical or other computational data. Measured data of potential utility to code validation efforts will also be considered for publication.

2. Code performance analysis. This usually involves identification of numerical accuracy or other limitations, solution convergence, numerical and physical modeling error, and parameter tradeoffs. However, it is also permissible to address issues such as ease-of-use, set-up time, run time, special outputs, or other special features.

3. Computational studies of basic physics. This involves using a code, algorithm, or computational technique to simulate reality in such a way that better or new physical insight or understanding is achieved.

4. New computational techniques, or new applications for existing computational techniques or codes.

5. "Tricks of the trade" in selecting and applying codes and techniques.

6. New codes, algorithms, code enhancement, and code fixes. This category is self-explanatory but includes significant changes to existing codes, such as applicability extensions, algorithm optimization, problem correction, limitation removal, or other performance improvement. **Note: Code (or algorithm) capability descriptions are not acceptable, unless they contain sufficient technical material to justify consideration.**

7. Code input/output issues. This normally involves innovations in input (such as input geometry standardization, automatic mesh generation, or computer-aided design) or in output (whether it be tabular, graphical, statistical, Fourier-transformed, or otherwise signal-processed). Material dealing with input/output database management, output interpretation, or other input/output issues will also be considered for publication.

8. Computer hardware issues. This is the category for analysis of hardware capabilities and limitations in meeting various types of electromagnetics computational requirements. Vector and parallel computational techniques and implementation are of particular interest.

Applications of interest include, but are not limited to, antennas (and their electromagnetic environments), networks, static fields, radar cross section, shielding, radiation hazards, biological effects, electromagnetic pulse (EMP), electromagnetic interference (EMI), electromagnetic compatibility (EMC), power transmission, charge transport, dielectric and magnetic materials, microwave components, MMIC technology, remote sensing and geophysics, communications systems, fiber optics, plasmas, particle accelerators, generators and motors, electromagnetic wave propagation, non-destructive evaluation, eddy currents, and inverse scattering.

Techniques of interest include frequency-domain and time-domain techniques, integral equation and differential equation techniques, diffraction theories, physical optics, moment methods, finite differences and finite element techniques, modal expansions, perturbation methods, and hybrid methods. This list is not exhaustive.

A unique feature of the Journal is the publication of unsuccessful efforts in applied computational electromagnetics. Publication of such material provides a means to discuss problem areas in electromagnetic modeling. Material representing an unsuccessful application or negative results in computational electromagnetics will be considered for publication only if a reasonable expectation of success (and a reasonable effort) are reflected. Moreover, such material must represent a problem area of potential interest to the ACES membership.

Where possible and appropriate, authors are required to provide statements of quantitative accuracy for measured and/or computed data. This issue is discussed in "Accuracy & Publication: Requiring quantitative accuracy statements to accompany data", by E.K. Miller, *ACES Newsletter*, Vol. 9, No. 3, pp. 23-29, 1994, ISBN 1056-9170.

EDITORIAL REVIEW

In order to ensure an appropriate level of quality control, papers are refereed. They are reviewed both for technical correctness and for adherence to the listed guidelines regarding information content. Authors should submit the initial manuscript in draft form so that any suggested changes can be made before the photo-ready copy is prepared for publication.

STYLE FOR CAMERA-READY COPY

The ACES Journal is flexible, within reason, in regard to style. However, certain requirements are in effect:

1. The paper title should NOT be placed on a separate page. The title, author(s), abstract, and (space permitting) beginning of the paper itself should all be on the first page. The title, author(s), and author affiliations should be centered (center-justified) on the first page.

2. An abstract is REQUIRED. The abstract should state the computer codes, computational techniques, and applications discussed in the paper (as applicable) and should otherwise be usable by technical abstracting and indexing services.

3. Either British English or American English spellings may be used, provided that each word is spelled consistently throughout the paper.

4. Any commonly-accepted format for referencing is permitted, provided that internal consistency of format is maintained. As a guideline for authors who have no other preference, we recommend that references be given by author(s) name and year in the body of the paper (with alphabetical listing of all references at the end of the paper). Titles of Journals, monographs, and similar publications should be in boldface or italic font or should be underlined. Titles of papers or articles should be in quotation marks.

5. Internal consistency shall also be maintained for other elements of style, such as equation numbering. As a guideline for authors who have no other preference, we suggest that equation numbers be placed in parentheses at the right column margin.

6. The intent and meaning of all text must be clear. For authors who are NOT masters of the English language, the ACES Editorial Staff will provide assistance with grammar (subject to clarity of intent and meaning).

7. Unused space should be minimized. Sections and subsections should not normally begin on a new page.

MATERIAL, SUBMITTAL FORMAT AND PROCEDURE

The preferred format for submission and subsequent review, is 12 point font or 12 cpi, double line spacing and single column per page. Four copies of all submissions should be sent to the Editor-in-Chief (see inside front cover). Each submission must be accompanied by a covering letter. The letter should include the name, address, and telephone and/or fax number and/or e-mail address of at least one of the authors.

Only camera-ready original copies are accepted for publication. The term **"camera-ready" means that the material is neat, legible, and reproducible**. The preferred font style is Times Roman 10 point (or equivalent) such as that used in this text. A double column format similar to that used here is preferred. **No author's work will be turned down once it has been accepted because of an inability to meet the requirements concerning fonts and format.** Full details are sent to the author(s) with the letter of acceptance.

There is NO requirement for India ink or for special paper; any plain white paper may be used. However, faded lines on figures and white streaks along fold lines should be avoided. Original figures - even paste-ups - are preferred over "nth-generation" photocopies. These original figures will be returned if you so request.

While ACES reserves the right to re-type any submitted material, this is not generally done.

PUBLICATION CHARGES

ACES members are allowed 12 pages per paper without charge; non-members are allowed 8 pages per paper without charge. Mandatory page charges of $75 a page apply to all pages in excess of 12 for members or 8 for non-members. Voluntary page charges are requested for the free (12 or 8) pages, but are NOT mandatory or required for publication. A priority courtesy guideline, which favors members, applies to paper backlogs. Full details are available from the Editor-in-Chief.

COPYRIGHTS AND RELEASES

Each primary author must sign a copyright form and obtain a release from his/her organization vesting the copyright with ACES. Forms will be provided by ACES. Both the author and his/her organization are allowed to use the copyrighted material freely for their own private purposes.

Permission is granted to quote short passages and reproduce figures and tables from an ACES Journal issue provided the source is cited. Copies of ACES Journal articles may be made in accordance with usage permitted by Sections 107 or 108 of the U.S. Copyright Law. This consent does not extend to other kinds of copying, such as for general distribution, for advertising or promotional purposes, for creating new collective works, or for resale. The reproduction of multiple copies and the use of articles or extracts for commercial purposes require the consent of the author and specific permission from ACES. Institutional members are allowed to copy any ACES Journal issue for their internal distribution only.